Calhoun: The NPS Institutional Archive
DSpace Repository

Theses and Dissertations 1. Thesis and Dissertation Collection, all items

1964

On-line program analysis and diagnosis.

Lightstone, John L.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/24787

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

http://www.nps.edu/library

NPS ARCHIVE
1964
LIGHTSTONE, J.

ON-LINE PROGRAM ANALYSIS AND DIAGNOSIS

JOHN L. LIGHTSTONE

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

Library
U. S. Naval Postgraduate School
Monterey, California

15

ON-LINE PROGRAM ANALYSIS AND DIAGNOSIS

by

John L. Lightstone

Lieutenant, United States Navy

Submitted in partial fulfillment of
the requirements for the degree of

MASTER OF SCIENCE
IN
ENGINEERING ELECTRONICS

United States Naval Postgraduate School
Monterey, California

1964

S Archive
4
ghtstone, J.

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY CA 93943-5101

Library
U. S. Naval Postgraduate School
Monterey, California

ON-LINE PROGRAM ANALYSIS
AND DIAGNOSIS

by

John L. Lightstone

This work is accepted as fulfilling
the thesis requirements for the degree of
MASTER OF SCIENCE
IN
ENGINEERING ELECTRONICS
from the
United States Naval Postgraduate School

ABSTRACT

For many program errors and program checkout problems, on-line techniques provide a promising method of attack. An approach developed in connection with time-sharing computer systems is examined. Then Program Trace, an on-line diagnostic program developed by the author for the CDC 1604 computer, and a Data Display Model DD 65 display and control console is presented and examined in detail.

TABLE OF CONTENTS

Section	Title	Page
1.	Introduction	1
2.	Checkout Problems and Program Errors	1
	2.1 Pre-compilation and Compilation Phase	2
	2.2 Running Phase	2
3.	Diagnostic Aids and Methods	4
4.	Weak Areas	5
5.	On-line Diagnostic Systems	6
6.	Computer Facilities	9
7.	Operation of Program Trace	14
8.	Trace Construction	23
9.	Conclusions	31
10.	Bibliography	34
Appendix I		
	DD 65 Display Word Format	35
	DD 65 Select and Sense Codes	36
Appendix II		
	Operational Register Display Flow Diagram	38
	Jump Table Tabulation Flow Diagram	39
Appendix III		
	Program Trace	40

LIST OF ILLUSTRATIONS

Figure Page

1. DD 65 Console 11

2. Display Tube Coordinates 12

3. Keyboards #1 and #2 13

4. Keyboard #2 16

5. Operational Registers Display 18

6. Jump Table Display 18

7. Simplified Flow Diagram of Trace 24

8. Flow Diagram of Trace 27

9. Keyboard #2 Portion of Keyboard Control 28

10. Keyboard #1 Portion of Keyboard Control 29

ABBREVIATIONS

A	Arithmetic Register
ADD	Execution Address
B^i	Index Register Number i
BBN	Bolt, Beranek and Newman, Inc.
BCD	Binary Coded Decimal
CDC	Control Data Corporation
DD	Data Display, Inc.
FC	Function Code
P	Program Address Register
Q	Auxiliary Arithmetic Register
SDC	Systems Development Corporation

1. Introduction.

Much has been done to automate the error detection processes for program checkout. On the other hand, it is apparent that a serious problem still exists. A significant portion of the time spent in the development of programs, both in terms of man hours and in terms of computer time, is in the checkout phase. Analysis of certain types of checkout problems is not easily automated. New approaches to the diagnostic problem are needed.

One promising approach is the use of the computer to provide on-line assistance to the programmer in analyzing his programs. In order to investigate the potentials and problems of this approach, an on-line diagnostic program was developed for use with the CDC 1604 computer and the DD 65 display console at the Naval Postgraduate School. Because of the technique used in the central part of the program, it is called Program Trace.

Before looking at Program Trace, it will be useful to catalog the types of checkout problems and program errors that occur and to list the aids and techniques available for attacking these problems, in order to determine the weak areas. Then a look will be taken at some on-line diagnostic programs, including a detailed examination of Program Trace.

2. Checkout Problems and Program Errors.

The checkout problems and program errors will be considered in two phases, those that occur or appear prior to or during compilation, and those that appear during the running of the program. Those in the first phase will be broken down into

clerical and syntactic errors, programming errors, and machine failures. Those in the running phase will be broken down into programming errors, conditional errors, and machine failures.

2.1 Pre-compilation and Compilation Phase.

The clerical and syntactic errors are the typographical mistakes, mispunched cards, and format errors which result in meaningless symbols or instructions.

Programming errors which appear during compilation are mistakes such as exceeding the storage capacity of the machine by reserving too much space for arrays or other data storage; not defining symbolic addresses; not identifying a reserved array; and calling for sub-programs not available on the system library.

Machine failures include anything from the dropping of a bit during transfer to or from magnetic tape to the failure of a logic component in the machine to the failure of a power supply. Also, included are failures of external equipment associated with the computer.

2.2 Running Phase.

Machine failures have no reference to the phase of the program. They are the same problems as already mentioned.

Conditional errors are those errors which depend on the status of certain parameters in the program or on the intermediate results of the computation, and are not always foreseeable. Examples are calculations resulting in a negative

argument for a square root function and the overflow of an arithmetic register during calculations.

Programming errors are flaws of a fundamental nature in the logical structure of the program and include a variety of problems. Three of these are the occurrance of unexpected jumps, deadends in loops, and programs that run to conclusion but with improper results.

The unexpected jumps are of two types - those that jump to an unused memory cell and stop the computer, and those that jump to another part of the object program or into the system programs and do not stop the computer immediately. In either case, it is difficult to determine where the jump originated.

Unending loops can occur from unindexed jumps or from conditional jumps for which the condition is never satisfied. In a dual instruction machine, such as the CDC 1604, a dead-end can also occur with a skip instruction in the lower half of a memory cell making continuous half exits on itself.

Programs which run to conclusion but with improper results are listed here not because they are a type of error of themselves, but because, quite often, this is the only external indication of internal programming errors. The cause can be an unexpected jump, as already mentioned, or a flaw in the logic used by the programmer in the construction of the program.

Other examples of programming errors are not supplying the required arguments for a sub-program, having input data for the program in improper format, and indexing a variable

array out of the storage area reserved for it in memory.

3. Diagnostic Aids and Methods.

The aids available during the pre-compilation and compilation phase include the error print-outs of the compiler and of the system executive program. The former giving indications of typographical and format errors and the latter giving indications of illegal procedures and some machine failures. These print-outs are well developed and will detect and pinpoint most of the clerical and syntactic errors.

Another valuable aid is a machine language or symbolic machine language listing of the object program which can be supplied by the compiler during compilation.

The diagnostic aids available during the running phase are generally limited to error print-outs by system library sub-programs when supplied with illegal arguments, and to console indications of the contents of the operational registers when the computer stops, console indications of registers overflowing, and console indications of hang-ups during input or output operations.

The diagnostic methods used during and after the running phase are generally variations of four techniques: The use of extra Print statements, Core Dumps, breakpoints, and stepping through the program.

By using extra Print statements at selected points in the object program, the value of parameters in the program can be determined at these points and an indication can be obtained of where trouble occurs.

The Core Dump is used to provide a printed record of the contents of the computer memory or selected portions of it for off-line analysis. This may be done at the completion of the run or at some intermediate point and in either a numeric or a mnemonic format.

The use of breakpoints to stop the computer at selected locations in the object program is valuable for examining and changing conditions at these points. However, it is not always known where it is desirable to stop.

In some cases where the trouble is localized or the action of a particular portion of the program is not understood, stepping through the program one instruction at a time is useful in order to follow the program execution exactly.

4. Weak Areas.

As mentioned, the diagnostic aids available during the compilation phase are well developed and are effective in detecting most of the errors which occur during that phase.

On the other hand, the diagnostic methods available in the running phase, while very useful, are often ineffective for determining the causes of some conditional errors or for locating errors in the logical structure of a program. As

a result, it is often necessary to use these methods in a series of repeated runs. The program is run; the programmer studies the results and makes changes and corrections; and another run is made, with the cycle repeated, perhaps many times until the errors are eliminated.

In a situation such as this, full use is not made either of the computer's potential or of the programmer's time, primarily because of the lack of communications between the computer and the programmer while the program is running. The computer does not know the programmer's intentions if they are not explicitly written into the program, and at other times, the programmer is limited in his communications with the computer.

5. On-line Diagnostic Systems.

The major work in developing on-line diagnostic programs has been in conjunction with the development of time-sharing computer systems. This is a natural result, since a major drawback of on-line systems is the inefficient use of computer time. With the ability provided by the time-sharing systems for several people to work on-line simultaneously, this drawback is eliminated.

Two time-sharing systems which have incorporated on-line diagnostic programs into the systems are those of Bolt, Berenac, and Newman [1,2] and the Systems Development Corporation [3]. The BBN system uses a Digital Equipment

Corporation PDP-1 computer and the SDC system uses an FSQ-32 computer with a PDP-1 for an input-output computer.

The operation of the two diagnostic programs is quite similar. Each operator who is running a program has a typewriter keyboard with which he can communicate with either his own program or with the time-sharing executive program and the diagnostic program (through the executive program). A prefix attached to the typed input indicates the program for which it is intended.

The two basic capabilities of the programs are the setting of breakpoints and the ability to examine and change the contents of memory cells, with variations available in how these are used.

Breakpoints (up to three at any one time) may be set by typing a short one or two character command followed by the breakpoint address. The break is accomplished by inserting a jump instruction at the desired break address which will jump into the diagnostic program. When the breakpoint is reached, the diagnostic program types a short message, including any information that may have been previously requested, on the typewriter, and waits for further instructions from the programmer.

Contents of memory cells, within the limits of memory storage used by the program, may be examined by typing another short command followed by the desired address.

Sections of memory of up to twenty cells or the operational registers may also be selected in this manner. Changes may be made in selected cells or registers by the use of another command followed by the new desired contents.

Breakpoints may be set or cells examined before the program is started, after it is finished, while it is stopped at a breakpoint, or while the program is running. In addition, through the executive program, the object program may be started, stopped, restarted, or started over again from the beginning.

The feature that makes these diagnostic programs so useful is their accessability and ease of use. They are always available to all users of the time-sharing system without the use of any special calling procedures. The diagnostic requests may be typed at any time, and they are sent directly to the diagnostic program by the system executive program.

There are limitations, however. In the case of problems such as unexpected jumps, there is no method for tracing or following the execution of the program as it is running.

A possible means of providing a tracing capability in these systems is by the following method. Provide an additional function in the diagnostic routine which would make a search of the object program before it is run. As a result of the search, the execution address of each jump instruction

in the object program would be changed to that of a position in a table which would be constructed in a reserved area at the end of the program. The table could be designed so that upon each jump into the table, the number of times that jump has been executed and/or the order in which the jumps have been executed could be recorded before a jump is made back to the originally assigned address in the object program.

There would be some problems in implementing this function, and it would make additional time and memory space requirements on the computer system, but it would be very useful for some problems. A special calling procedure would have to be provided for the jump table function, since it could not be as freely accessable as the rest of the diagnostic program.

Program Trace is a program, written by the author, designed to provide on-line diagnostic functions, including a tracing function in a non time-sharing system. However, before discussing Program Trace, a description of the facilities will be provided.

6. Computer Facilities.

The computer facilities at the Naval Postgraduate School include a CDC 1604 computer [6] and a CDC 160 computer in the school's computer center and a Data Display model DD 65 display console [8] and another CDC 160 computer in the Digital Control Laboratory.

The CDC 1604 is a 48 bit, dual instruction computer with a 32K core memory. It has one high speed unbuffered input-output channel and three buffered input and three buffered output channels.

The Fortran 60 System [7] is the Fortran processor in use with the CDC 1604. When the processor is loaded in memory, the resident program occupies the first 4000_8 cells, and the compiler occupies the next 14000_8 cells, so that as each object program is loaded into memory, it begins at address 20000_8. Any library sub-programs which are called are loaded at the end of the object program. The Fortran compiler is designed for either normal Fortran coding or symbolic machine language coding, or for intermixing of the two.

The two CDC 160 computers are 12 bit, single address machines, with 4K core memories, of which only the first 64 cells are directly addressable. They each have one input and one output channel.

The DD 65 is a display and control console, especially configured by Data Display for the Naval Postgraduate School. The console is shown in Figure 1. The associated core memory and logic are in a separate cabinet.

The console has two 12" cathode ray tubes for display of alpha-numeric and vector symbols. For other applications, radar video may also be displayed on the left tube. Display

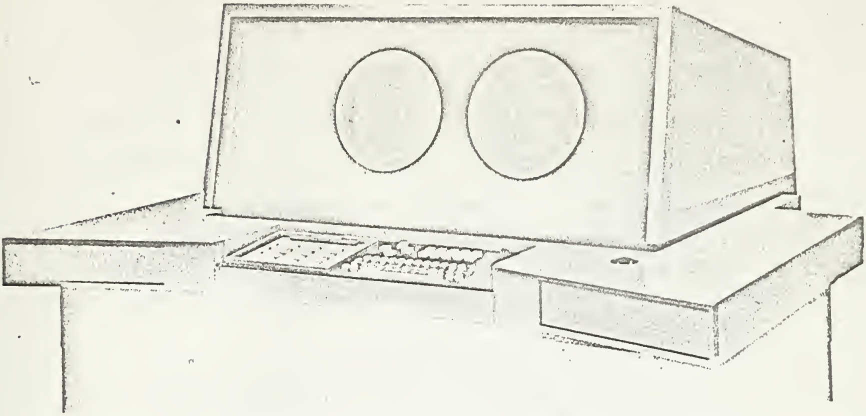


Figure 1. DD 65 Console

commands are stored in the 512 word addressable core memory. The commands are 48 bit words which each correspond to one CDC 1604 word or four CDC 160 words.

The contents of the memory are scanned and displayed approximately 20 times a second to provide a flicker free presentation. The memory can be updated by either the CDC 1604 or the CDC 160 (but not by both at the present time) at any time during the scan. A complete read-write memory cycle is 12.8 micro-seconds.

The position of a character on the screen is designated in X-Y coordinates with the origin at the center of the screen.

As shown in Figure 2, the X (Y) coordinate is given as a nine bit number which increases from 000 at the origin to 377 at the right (top) edge, and from 400 at the left (bottom) edge to 777 at the origin. After an initial position is designated by the use of a designator word, successive characters may be incremented either horizontally or vertically from that position by the use of string words. The formats for designator and string words are shown in Appendix I.

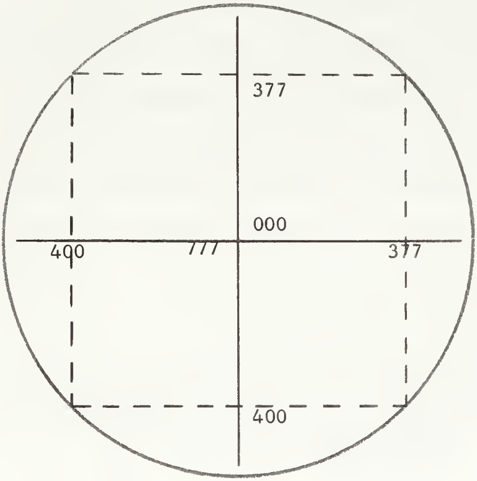


Figure 2. Display Tube Coordinates

The keyboards, Figure 3, are used to provide outputs to the computer. Since there is no direct connection with the display generation circuits or display memory, all control and display functions are performed through the computer. The typewriter keyboard is referred to as keyboard #1, and all other keys and buttons are referred to as keyboard #2.

Cables and a switch on the DD 65 memory cabinet are provided so that the display may be operated with either the CDC 1604 or with the CDC 160 in the Digital Control Laboratory. In the latter case, the CDC 160 may be operated independently or as a satellite of the CDC 1604. For satellite operations, communications between the CDC 160 and the CDC 1604 are through one set of the CDC 1604's buffered input-output channels. When the DD 65 is operated directly with the CDC 1604, communications are via the high speed input-output channel of the CDC 1604.

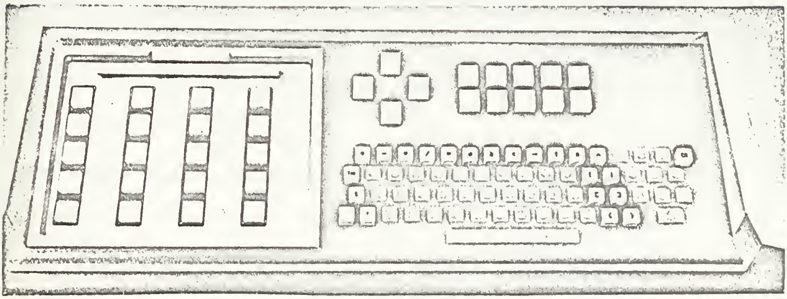


Figure 3. Keyboards #1 and #2.

In considering which configuration to use in implementing Program Trace, both the direct connection of the display to the CDC 1604 and the satellite arrangement have advantages. In the latter case, a load is taken off of the CDC 1604 by having the display generation programs located in the CDC 160. Also, this configuration might more easily be combined with other systems in the future. On the other hand, the direct connection is easier to implement initially. For this reason, the direct connection was chosen.

7. Operation of Program Trace.

Program Trace is written in the form of a sub-program. It is designed so that it can be made a part of the resident library and be callable by the object program (the program to be diagnosed). At present, it is loaded at the end of the object program.

Program Trace is intended for the analysis of machine coded programs. However, it can be useful for diagnosis of Fortran coded programs if a machine code listing of the program is obtained. The Fortran compiler can provide such a listing.

One argument must be provided when Program Trace is called, and that is the address of the instruction immediately following the calling statement. This is done by the use of two symbolic statements prior to the Fortran calling statement.

```
ENA (L+3)
STA (ADDRESS)
CALL TRACE (ADDRESS)
```

These statements are normally placed near the beginning of the object program, although they may be placed at other positions if only the latter portion of the program is to be traced.

When the program is run, the Trace program takes control of the execution and first checks to see if the DD 65 is available and ready. If not, the object program is run to completion under the control of Trace, and a table is prepared of each jump instruction executed in the object program including the address of the jump instruction, the address to which it jumped, and the number of times it was executed. The table is printed out in blocks of 50 jump instructions as the table is filled or at the end of the program.

If the DD 65 is available and ready, then its memory will be erased, its keyboard lights set to correspond with initial Trace control flag settings, and an introduction will be displayed on the left tube. The introduction will give the location of the object program in memory, and by giving the location of Trace, will indicate where the object program ends.

```
PREAMBLE OF YOUR PROGRAM BEGINS AT CELL  20000
INSTRUCTION AFTER 'CALL TRACE' IS CELL   20006
SUBROUTINE TRACE BEGINS AT CELL          21703
```

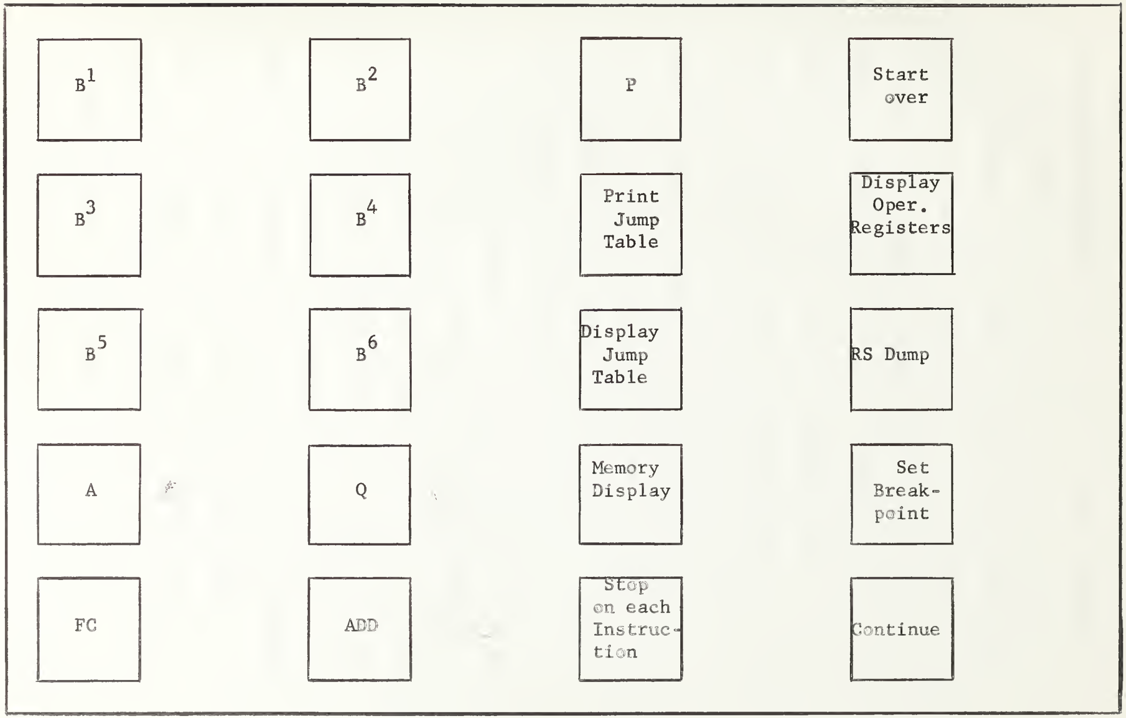
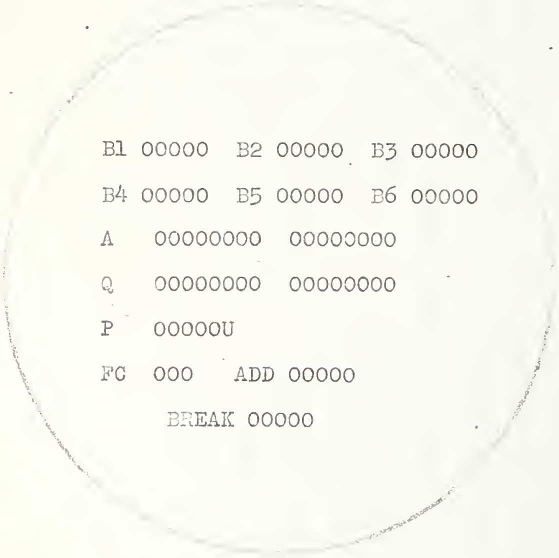


Figure 4. Keyboard #2

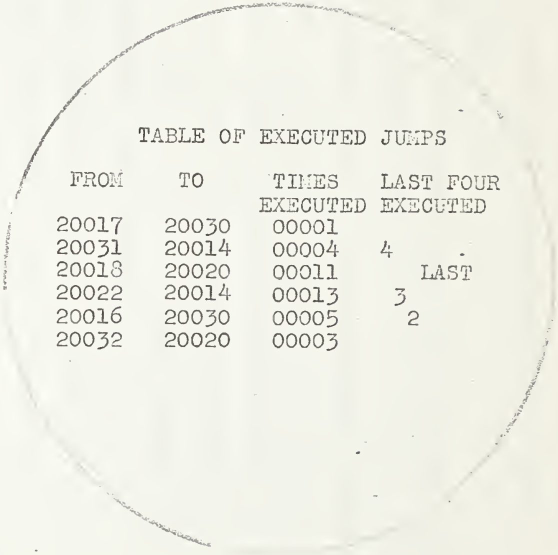
After displaying the introduction, Trace will wait for the programmer to take control at the DD 65 keyboard by pressing the CONTINUE button, which is shown in Figure 4. When the CONTINUE button is pressed, the object program will be allowed to run until the first jump instruction is reached. At that time, the contents of the operational registers will be displayed on the left tube (Figure 5), and the headings for the jump table will be displayed on the right tube (Figure 6). Each time the CONTINUE button is pressed, the object program will advance to the next jump instruction, and the operational register and jump table displays will be updated. The fourth column of the jump table will indicate the last four jump instructions executed in the object program. If an unexpected jump occurs, the table will show where the jump originated and the sequence that led up to it.

Keyboard #2 (Figure 4) provides the ability to change the contents of the operational registers, to examine the contents of memory, to record changes made in the program, to select and deselect various displays, to change the degree of control of program execution, and to restart the object program. The various functions are selected by pressing the desired button and deselected by pressing the button a second time. The keyboard light associated with the button indicates whether it is selected or deselected. Functions which might be conflicting are automatically deselected when a new function is selected. The functions which are initially selected



```
B1 00000   B2 00000   B3 00000

B4 00000   B5 00000   B6 00000

A    00000000   00000000

Q    00000000   00000000

P    00000U

FC   000    ADD 00000

        BREAK 00000
```

Figure 5. Operational Registers Display



```
         TABLE OF EXECUTED JUMPS

 FROM     TO       TIMES     LAST FOUR
                 EXECUTED  EXECUTED
20017    20030    00001
20031    20014    00004     4
20018    20020    00011        LAST
20022    20014    00013      3
20016    20030    00005       2
20032    20020    00003
```

Figure 6. Jump Table Display

when the program is started are PRINT JUMP TABLE, DISPLAY JUMP TABLE, DISPLAY OPERATIONAL REGISTERS, and CONTINUE (which is always active).

Control of the execution of the object program is exercised by the CONTINUE button as previously described. Variations are possible by the use of the STOP ON EACH INSTRUCTION and SET BREAKPOINT functions. With the STOP ON EACH INSTRUCTION function selected, the object program is stopped prior to the execution of each instruction and the displays updated. This provides the ability to step through portions of the program where closer examination is desired.

When the BREAKPOINT function is selected, the normal stops at each jump instruction are deleted; and when the CONTINUE button is pressed, the program runs continuously without stopping until the breakpoint address is reached or until the program is completed. The latter case will be discussed in a moment. Note that the program may be deliberately run to completion with no stops by selecting the BREAKPOINT function, but with the breakpoint address left zero.

The breakpoint address is set by first selecting the function and then typing the address on keyboard #1. The address will appear at the bottom of the operational register display as it is typed. If too many digits are typed, the address will be reset to zero and the breakpoint function deselected.

It should be noted that the octal number system is used for all displays and keyboard inputs.

The MEMORY DISPLAY function provides a means of inspecting constants or other desired portions of the object program. The function displays a block of twenty-one memory cells and may be selected at any time. After it is selected, the address of the first cell is typed on keyboard #1. This address will appear above the operational register display on the left tube. As the last digit of the address is typed, the operational register display will be cleared and the contents of that cell and the twenty successive memory cells will be displayed in its place. When the MEMORY DISPLAY is deselected, the operational register display is restored.

The DISPLAY JUMP TABLE and DISPLAY OPERATIONAL REGISTERS buttons provide the option of deleting the respective displays. Some computing time is saved by skipping these display generation routines. However, the time is normally insignificant compared with the dead time waiting for keyboard inputs, since the displays are only generated prior to a stop. The PRINT JUMP TABLE button can be used to delete the printing of a hard copy of the jump table at the end of the program. If both jump table functions are deleted, the table is not compiled. In this case, a significant amount of time is saved during continuous runs with the breakpoint set.

The left two columns on keyboard #2 and the button marked P are the register change functions. In each case, the desired register is selected and then the new contents for the register are typed on keyboard #1. As the contents are typed, they will be displayed on the screen. However, the format for typing is different for some of the registers.

For the index registers, B1 through B6, the number is typed in a normal manner from left to right, except that zeros at the lower end of the number need not be typed. For example, to put the number 02300 in one of the index registers, either 02300 or 023 may be typed. The function is deselected by pressing the selected button a second time or by striking a sixth key on keyboard #1.

For the A and Q registers, the procedure is slightly different. Quite often, the numbers desired in these registers amount to only a few digits in the lower end. Therefore, the order is reversed so that numbers are typed from right to left and zeros in upper portion of the register need not be typed. For example, to put the number 00000000 00002300 in the A register, the function is selected and then the number 0032 is typed on keyboard #1. To deselect the function, the A button is pressed a second time, or a seventeenth key is struck on keyboard #1.

The use of the function code, FC, and execution address, ADD, change functions are quite similar to that of the index registers. For the program address, P, register,

however, the full five digit address (including zeros) must be typed, and then a U or an L typed at the end to indicate either the upper or lower half of the address. As the new program address is typed, it will appear on the operational register display, and as the U or the L is typed, the remainder of the operational register display will change to correspond with the new program address.

The function code, execution address, and program address change functions should be used very carefully since they can disrupt the execution of the object program. The program address function, however, can be very useful for repeating or skipping sections of the object program.

The START OVER function provides an easy method of restarting the object program from the beginning. It may be used when the program has run to completion or at any intermediate time. To prevent accidental restarts, a two step operation is required. First, the button is pressed, and then any key on keyboard #1 is struck.

RSDUMP is a sub-program available on the system library. It provides a hard copy output of the object program in an octal format with mnemonics. When the RSDUMP button is pressed, the associated keyboard light will come on and remain on until the output is completed, and then the light will go out.

In the discussion of the keyboard functions, it should

be noted that the functions can only be used while execution of the object program is stopped, and that none of the functions except START OVER and P, will advance the object program until the CONTINUE button is pressed.

When the object program is run to completion, the word END will appear in the center of the left tube. All of the keyboard functions will be available at this time. However, the ones most likely to be used are MEMORY DISPLAY, to examine final values in the program; RSDUMP, to make a hard copy of the object program; or START OVER, to begin again. When the programmer is finished, the CONTINUE button is depressed to give control back to Monitor.

8. Trace Construction.

Figure 7 shows a simplified flow diagram of Trace with some functions omitted and with conditions as set initially in the program. The two loops in the program are clearly evident; one loop for jump instructions, and a second loop for the other instructions.

The program is entered initially from the calling statement in the object program. Trace then proceeds by pulling successive instructions out of the object program, one at a time. The type of instruction is determined by comparison with a table, and the instruction is then stored in the upper half (no matter whether it was an upper or lower half instruction) of a reserved cell

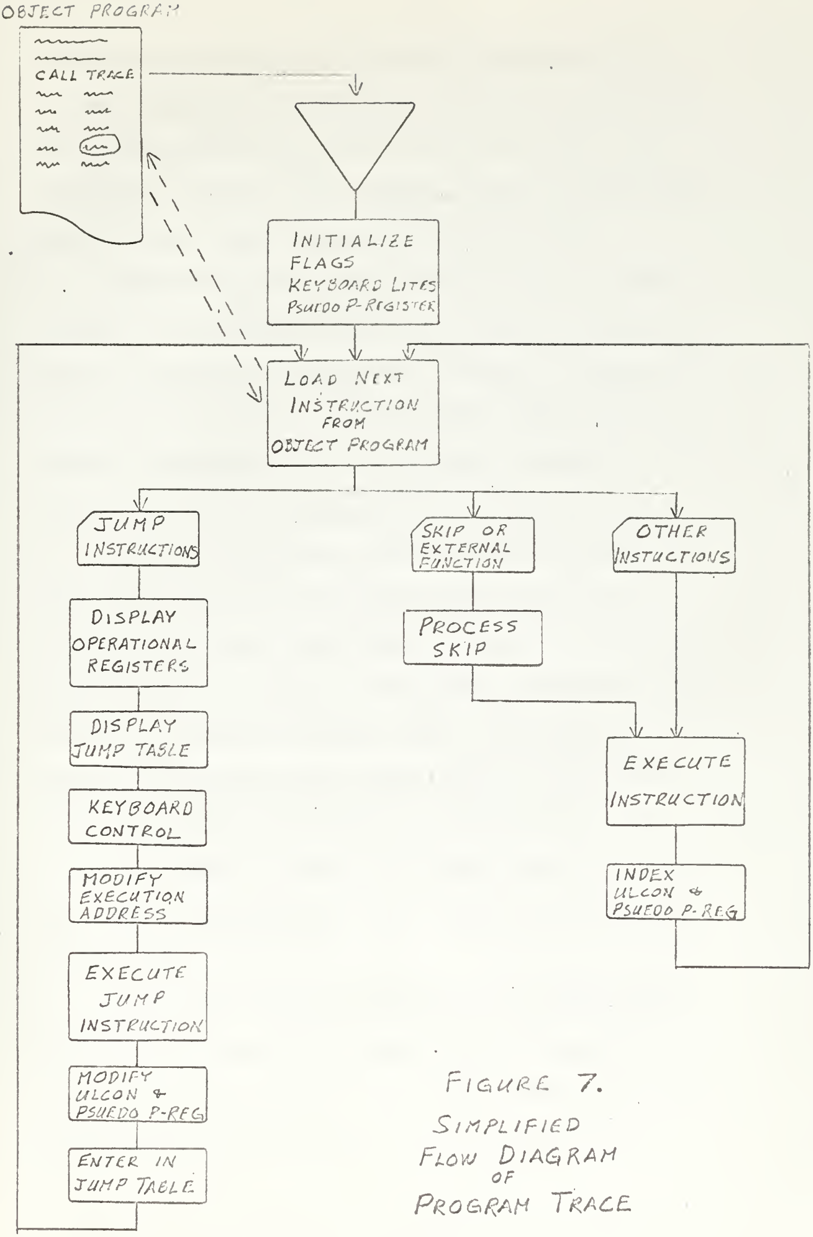


FIGURE 7.

SIMPLIFIED
FLOW DIAGRAM
OF
PROGRAM TRACE

either in the jump loop or the other instructions loop to await execution. An exception is that lower half skip and external function instructions are retained in the lower half, and some slight changes, indicated by the box marked Process Skip, are made to accommodate them.

Instructions other than jump instructions are then executed, and the upper-lower counter (ULCON), which keeps track of whether the instructions are upper or lower half instructions, and the psuedo P-register (PREG) are incremented as necessary. The loop is then completed by returning to load the next instruction from the object program.

For jump instructions, the operational register and jump table display routines and keyboard control are included in the loop. Also, before the jump instruction can be executed, its execution address must be modified so that control is retained in Trace. After the instruction is executed, the upper-lower counter (ULCON) and the psuedo P-register (PREG) are modified to correspond with the execution address and the type of jump (normal or return). The jump is then tabulated in the jump table and the loop is completed.

Although not shown in Figure 7, the display routines and keyboard control are bypassed when the breakpoint is set or when the DD 65 is not available. Also, the displays may be selectively bypassed by use of Keyboard Control.

In Figure 8, the flow diagram is expanded to show some additional functions and variations. For example, the display routines and Keyboard Control can be entered from the other instruction loop when the STOP ON EACH INSTRUCTION function is selected or when a breakpoint is reached. Also, the execution address of each jump is examined after it is tabulated in the jump table, and a jump to Monitor is used as the indication that the program is completed. If the program is completed, the jump table is printed, the word END is displayed, and Keyboard Control is entered.

Flow diagrams for Keyboard Control are shown in Figures 9 and 10. Keyboard Control is entered from one of three points marked by a, b, and c in Figure 8. Exit is by way of the CONTINUE button shown in Figure 9 and is always to the point from which the routine was entered. Keyboards #1 and #2 are alternately sensed until one or the other is hit. Figure 9 is the kayboard #2 portion of Keyboard Control, and Figure 10 is the keyboard #1 portion.

When keyboard #2 is hit, the input is decoded to determine which button was hit, and the function is selected or deselected by shifting a flag which is alternately positive and negative. If the function is selected, a return jump is made to the Master Deselect routine, which clears the flags and keyboard lights of conflicting functions. Those functions which have no conflicts do not use the Master Deselect routine.

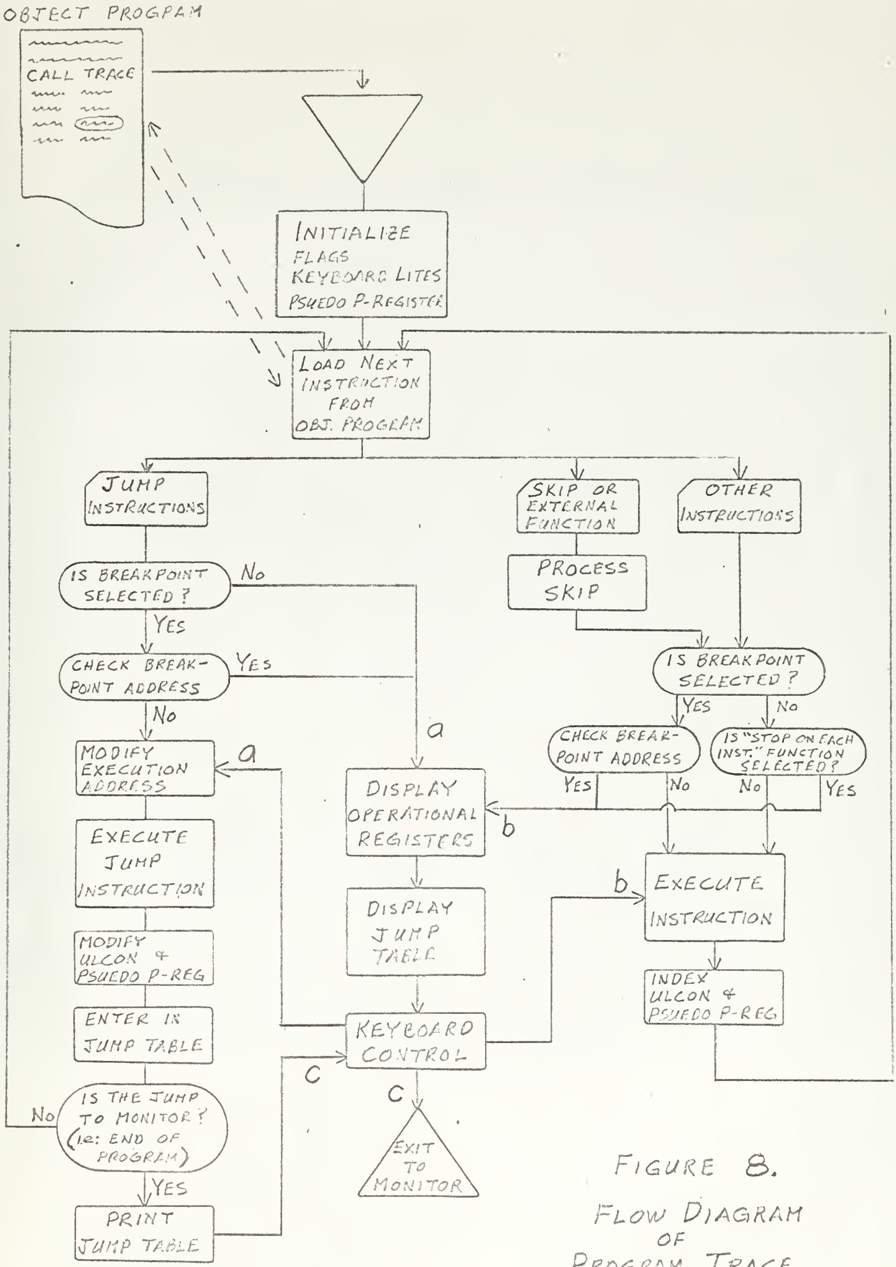


FIGURE 8.

FLOW DIAGRAM
OF
PROGRAM TRACE

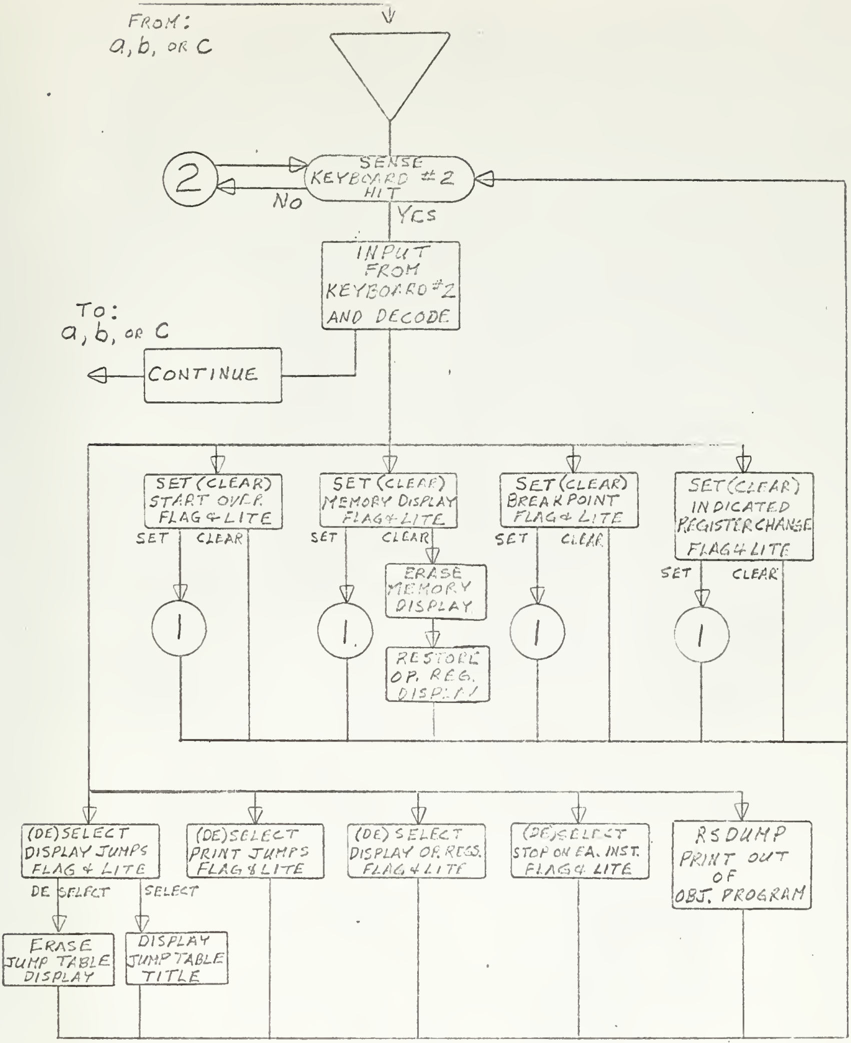
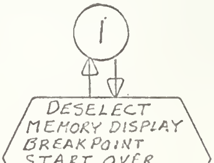

FROM:
a, b, OR C
2
SENSE KEYBOARD #2 HIT
No
Yes
INPUT FROM KEYBOARD #2 AND DECODE
To:
a, b, OR C
CONTINUE
SET (CLEAR) START OVER FLAG & LITE
SET
CLEAR
1
SET (CLEAR) MEMORY DISPLAY FLAG & LITE
SET
CLEAR
1
ERASE MEMORY DISPLAY
RESTORE OP. REG. DISPLAY
SET (CLEAR) BREAK POINT FLAG & LITE
SET
CLEAR
1
SET (CLEAR) INDICATED REGISTER CHANGE FLAG & LITE
SET
CLEAR
1
(DE) SELECT DISPLAY JUMPS FLAG & LITE
DE SELECT
SELECT
ERASE JUMP TABLE DISPLAY
DISPLAY JUMP TABLE TITLE
(DE) SELECT PRINT JUMPS FLAG & LITE
(DE) SELECT DISPLAY OP. REGS. FLAG & LITE
(DE) SELECT STOP ON EA. INST. FLAG & LITE
RS DUMP PRINT OUT OF OBJ. PROGRAM

FIGURE 9.

KEYBOARD #2

PORTION OF KEYBOARD CONTROL


1
DESELECT MEMORY DISPLAY BREAKPOINT START OVER REGISTER CHG. FNS ASSOCIATED LITES

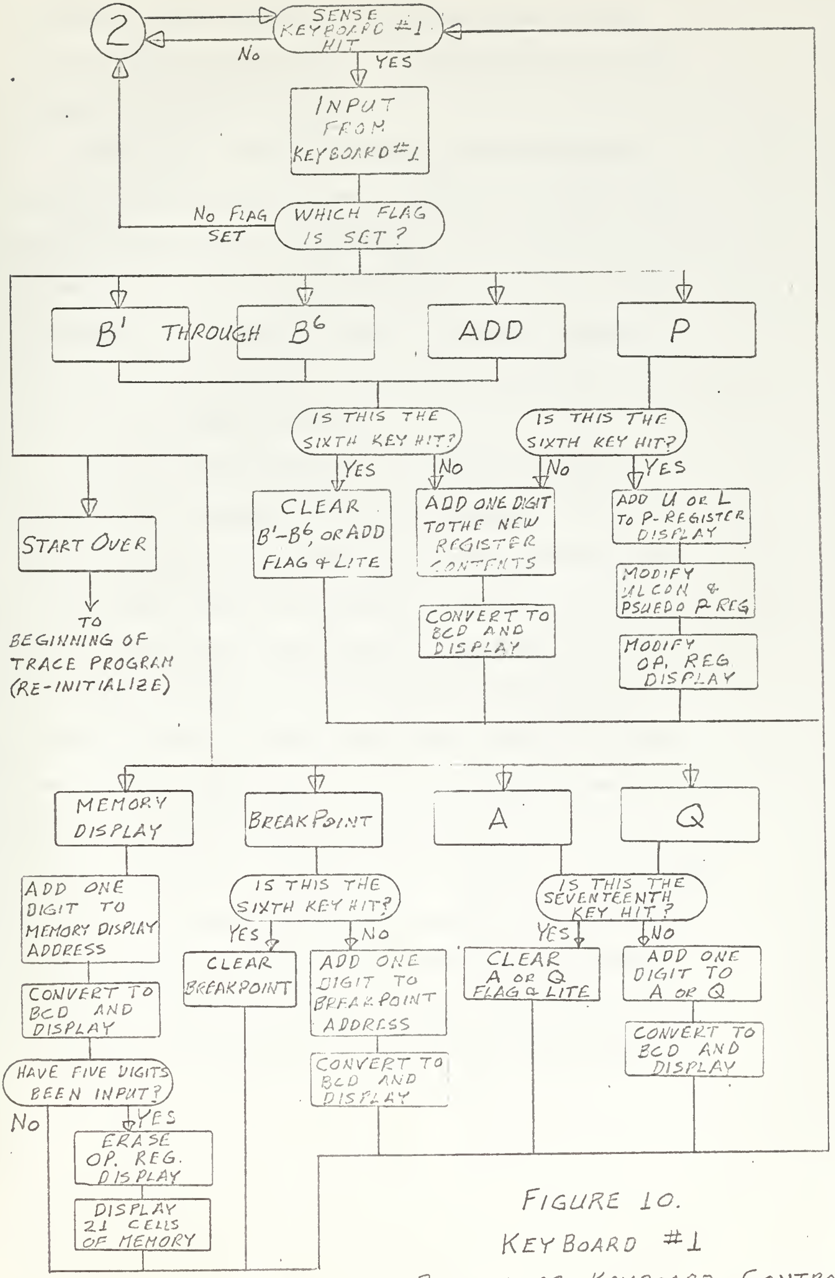

2
SENSE KEYBOARD #1 HIT
No
YES
INPUT FROM KEYBOARD #1
No FLAG SET
WHICH FLAG IS SET?
B¹ THROUGH B⁶
ADD
P
IS THIS THE SIXTH KEY HIT?
YES
No
IS THIS THE SIXTH KEY HIT?
No
YES
START OVER
TO BEGINNING OF TRACE PROGRAM (RE-INITIALIZE)
CLEAR B¹-B⁶, OR ADD FLAG & LITE
ADD ONE DIGIT TO THE NEW REGISTER CONTENTS
CONVERT TO BCD AND DISPLAY
ADD U OR L TO P-REGISTER DISPLAY
MODIFY ULCON & PSUEDO P REG
MODIFY OP. REG DISPLAY
MEMORY DISPLAY
BREAKPOINT
A
Q
ADD ONE DIGIT TO MEMORY DISPLAY ADDRESS
CONVERT TO BCD AND DISPLAY
HAVE FIVE DIGITS BEEN INPUT?
No
YES
ERASE OP. REG. DISPLAY
DISPLAY 21 CELLS OF MEMORY
IS THIS THE SIXTH KEY HIT?
YES
No
CLEAR BREAKPOINT
ADD ONE DIGIT TO BREAKPOINT ADDRESS
CONVERT TO BCD AND DISPLAY
IS THIS THE SEVENTEENTH KEY HIT?
YES
No
CLEAR A OR Q FLAG & LITE
ADD ONE DIGIT TO A OR Q
CONVERT TO BCD AND DISPLAY

FIGURE 10.

KEYBOARD #1

PORTION OF KEYBOARD CONTROL

As shown in the diagram for the Memory Display and Jump Table Display functions, there are some additional display generation and erase features.

The register change functions, the Memory Display function, the Breakpoint function, and the Start Over function require inputs from keyboard #1. The keyboard #1 hits, which are in the form of a six bit BCD code, are input one at a time, and the control flags are checked to determine for which function they are intended. If no control flags are set, the input is discarded.

With two exceptions, the only inputs used are the octal digits 0 through 7. The first exception is that the Start Over function only checks that there has been a keyboard #1 hit. The second exception is the U or L needed by the Program Address change function. Otherwise, if any of the upper three bits of the BCD input are not zero, the input is treated as a zero. The reason being that the BCD codes for the other digits are just the digits themselves. The processing of the inputs by the various functions is shown in Figure 10. As each digit is received by the selected function, the desired number is assembled, both in binary form for use in the program and in BCD format for display. A count is kept of the number of digits input in order to determine when to deselect or execute the selected function.

Two routines are used for this purpose. A "48 bit registers" routine handles the 16 digit inputs for the A and Q registers, and a "15 bit registers" routine processes the five digit inputs for the other functions. When processing of a digit is completed, a return is made to the keyboard sensing wait loop to await the next keyboard hit.

Flow diagrams for other sections of Trace are included in Appendix II.

9. Conclusions.

The primary limitation of Program Trace is time, in terms of both running time and dead time. To calculate the running time for the program, consider that the breakpoint is set, the program is running, and the breakpoint has not yet been reached. Each non-jump, non-skip instruction in the object program requires the execution of from 43 to 45 instructions, and each skip or external function instruction requires the execution of from 43 to 48 instructions by Trace. For conditional jump instructions which are not satisfied, the number of instructions executed is 41. Executed jumps which are not a part of the object program require 59 instructions. These are jumps executed in library subprograms. They are executed under Trace control, but are not tabulated in the jump table.

For jump instructions in the object program, 86 to 101 instructions are executed by Trace. This includes the jump table tabulations and the end-of-program check, but it does not include display generation.

Thus, for example, an object program consisting of one-fifth jump instructions would have a running time of approximately 55 times normal when it is under Trace control with no display generation.

The display generation routines which are only executed before a stop require the execution of several hundred instructions. However, this time is negligible compared with the dead time waiting for keyboard inputs. Likewise, some of the Keyboard Control functions are relatively lengthy but their running time is small compared with the dead time. As long as the dead time is not utilized for other purposes, the length of the display routines and Keyboard Control is not important.

Although Program Trace was specifically designed for a non time-sharing system, it is constructed so that it could be incorporated into a simple time-sharing arrangement. For example, an area of computer memory could be reserved for Program Trace. The keyboard sensing wait loop, in which first one and then the other keyboard is checked until there is a hit, could be replaced by interrupts. Then, during the normal dead time, the computer could continue with other

jobs until interrupted by a keyboard hit. Since, in this arrangement, computer running time would be more important, it might also be desirable to transfer the display generation and Keyboard Control routines to the CDC 160 in order to take some of the load off of the CDC 1604.

However, considering only the non time-sharing case, Program Trace can be very useful in many cases where the normal diagnostic methods become ineffective. The auto-monitoring of program execution can well be worth the investment of some additional computer time. Not only can an overall gain in efficiency of program diagnosis and checkout be realized, but in many cases, there will be no loss in computer time, since the need for repeated runs will be eliminated.

BIBLIOGRAPHY

1. Bolt, Beranek and Newman, Inc. A time-sharing debugging system for a small computer, by J. McCarthy, S. Boilen, E. Fredkin, and J. C. R. Licklider.

2. Bolt, Beranek and Newman, Inc. DEC debugging tape program write-up. Digital-1-3-5.

3. Systems Development Corporation. Time-sharing system user's guide. Tech Memo 1354/000/00, 7 Nov. 1963.

4. Tsui, F. F. A flexible and inexpensive method of monitoring program execution in a digital computer. IRE transactions on digital computers, V. EC-10, June 1961: 253-259.

5. Barron, D. W. and D. F. Hartley. Techniques for program error diagnosis on EDSAC 2. The (British) Computer Journal, V. 6, April 1963: 44-49.

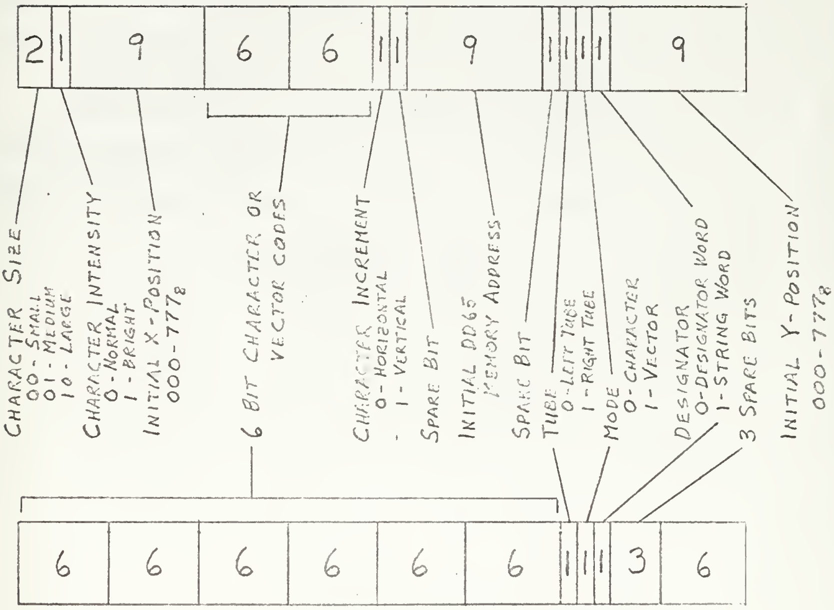
6. Control Data 1604 Computer Programming Manual. CDC publication #167.

7. Fortran system for the control data 1604 computer. CDC publication #087A.

8. Data Display, Incorporated. Data Display Model DD 65 instruction manual.

APPENDIX I

DD 65 DESIGNATOR WORD FORMAT



DD 65 STRING WORD FORMAT

DD 65 EXTERNAL FUNCTION CODES

Function	1604 Select Code
Select Keyboard 1 for Input	C7140
Keyboard 2 for Input	C7120
Select Track Ball "X"	C7102
Track Ball "Y"	C7104
Select Range Switch	C7110
Select Memory Update from 1604	C7010
Select Radar Target Data to 1604	C7002
Select Interruption:	
Keyboard 1 Hit	C7105
Keyboard 2 Hit	C7103
Radar Pulse Hit	C7141
Release Interrupt Request	C7111
Remove all Interrupt Selects	C7121

C = 1604 Data Channel Number

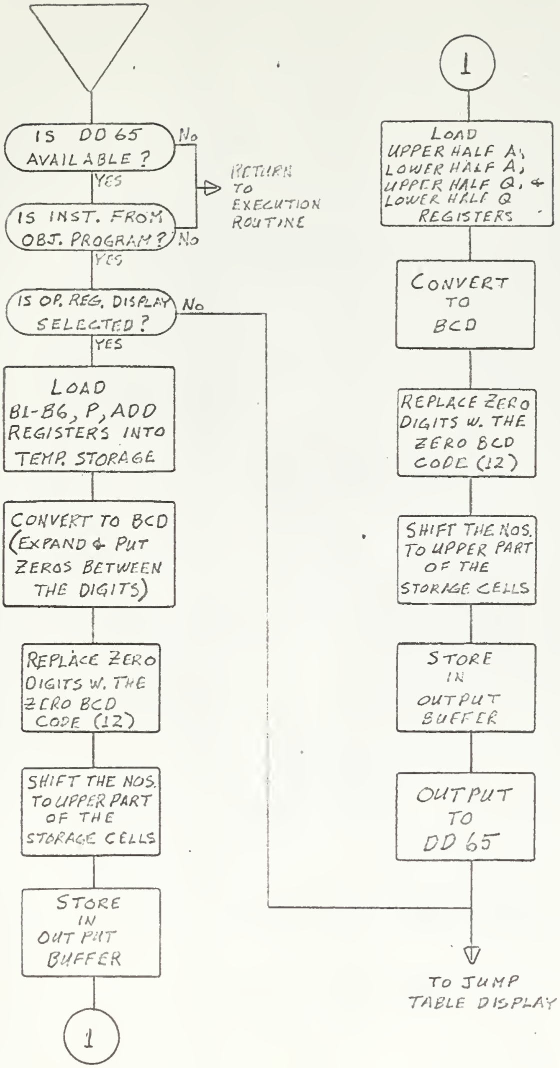
1604 SENSE CODES

Condition	Sense Code *
Keyboard 1 Hit	C7172
Keyboard 2 Hit	C7175
Keyboard 1 Not Hit	C7173
Keyboard 2 Not Hit	C7174
Tab Not Hit	C7157
Carriage Return Not Hit	C7166
Keyboard 1 Not Selected	C7167
Keyboard 2 Not Selected	C7037
DD 65 Interrupt	C7156
DD 65 from 1604 Selected	C7010
Radar to 1604 Selected	C7002

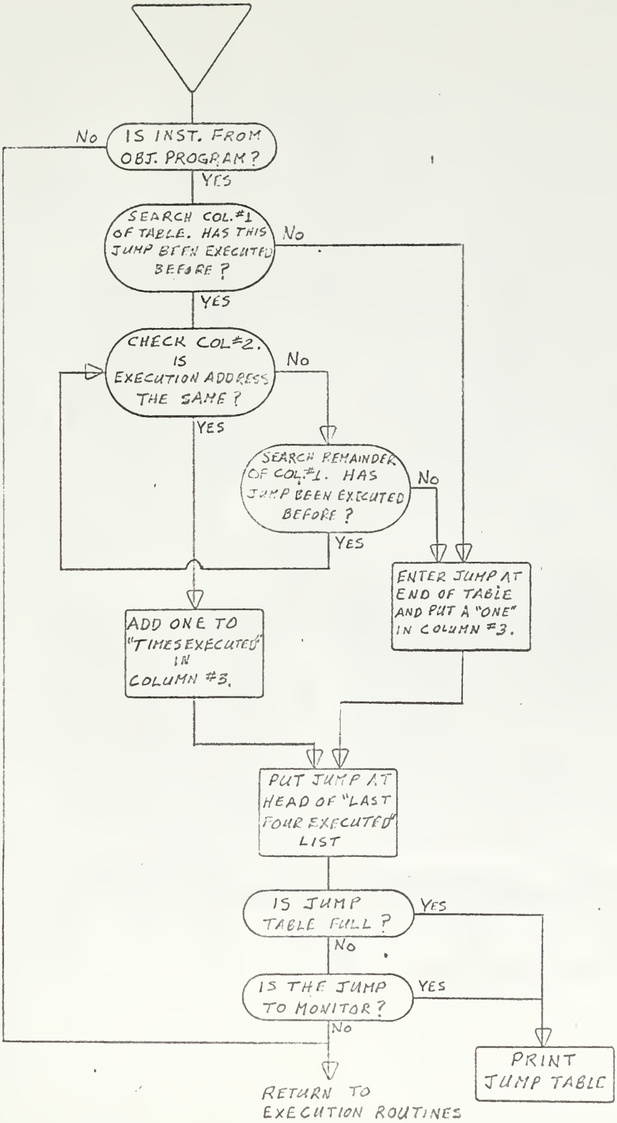
C = 1604 Data Channel Number

* Full Exit on a Positive Response.
Half Exit on a Negative Response.

APPENDIX II


IS DD 65 AVAILABLE ?
No
YES
IS INST. FROM OBJ. PROGRAM ?
No
YES
RETURN TO EXECUTION ROUTINE
IS OP. REG. DISPLAY SELECTED ?
No
YES
LOAD B1-B6, P, ADD REGISTERS INTO TEMP. STORAGE
CONVERT TO BCD (EXPAND & PUT ZEROS BETWEEN THE DIGITS)
REPLACE ZERO DIGITS W. THE ZERO BCD CODE (12)
SHIFT THE NOS. TO UPPER PART OF THE STORAGE CELLS
STORE IN OUT PUT BUFFER
1
1
LOAD UPPER HALF A, LOWER HALF A, UPPER HALF Q, & LOWER HALF Q REGISTERS
CONVERT TO BCD
REPLACE ZERO DIGITS W. THE ZERO BCD CODE (12)
SHIFT THE NOS. TO UPPER PART OF THE STORAGE CELLS
STORE IN OUTPUT BUFFER
OUTPUT TO DD 65
TO JUMP TABLE DISPLAY

OPERATIONAL REGISTER DISPLAY


No
IS INST. FROM OBJ. PROGRAM?
YES
SEARCH COL.#1 OF TABLE. HAS THIS JUMP BEEN EXECUTED BEFORE?
No
YES
CHECK COL#2. IS EXECUTION ADDRESS THE SAME?
No
YES
SEARCH REMAINDER OF COL#1. HAS JUMP BEEN EXECUTED BEFORE?
No
YES
ADD ONE TO "TIMES EXECUTED" IN COLUMN #3.
ENTER JUMP AT END OF TABLE AND PUT A "ONE" IN COLUMN #3.
PUT JUMP AT HEAD OF "LAST FOUR EXECUTED" LIST
IS JUMP TABLE FULL?
YES
No
IS THE JUMP TO MONITOR?
YES
No
PRINT JUMP TABLE
RETURN TO EXECUTION ROUTINES

JUMP TABLE TABULATION

APPENDIX III   PROGRAM TRACE

```
      SUBROUTINE TRACE(ADDRESS)

      DIMENSION BUF1(50),BUF2(50),BUF3(50)
      CON(MSK1 =0000000077777777B,  MSK2 =7770000000000000B,
     *    MSK3 =2040000000000000B,  MSK4 =7777777700000000B,
     *    MSK5 =0000000000077777B,  MSK6 =0000000000070000B,
     *    MSK7 =0000770000000000B,  MSK8 =0000000070000000B,
     *    MSK9 =7700000000000000B,  MSK10=7777000077777777B,
     *    MSK11=7777000000000000B,  MSK12=0070000000000000B,
     *    MSK13=7777000077777000B,  MSK14=0000000007770000B,
     *    MSK15=0000000077770000B)
      CON(PASS =5000000000000000B,  ULCON=5252525252525252B,
     *    ZERO =1212121212121212B,  LOW  =0000000000430000B,
     *    UP   =0000000000240000B,  BK   =1212121212000000B,
     *    END  =6545640000000000B,  END1 =5720000016321760B,
     *    ERASE=0000000000001000B)
*     OPERATIONAL REGISTERS OUTPUT BUF

      CON(DDTB  =  0012121212120000B,
     1    BR1   =  5400620100001240B,
     2    BR2   =  0012121212120000B,
     3    BR2   =  5660620200041240B,
     4    BR3   =  0012121212120000B,
     5    BR3   =  5140620300101240B,
     6    BR4   =  0012121212120000B,
     7    BR4   =  5400620400141140B)
      CON(BR5   =  0012121212120000B,
     1    BR5   =  5660620500201140B,
     2    BR6   =  0012121212120000B,
     3    BR6   =  5140620600241140B,
     4    AR    =  5400610000301040B,
     5    QR    =  5400500000421740B,
     6    ARL1  =  1212121212120000B,
     7    ARL   =  5500121200321040B)
      CON(ARR   =  1212121212120000B,
     1    ARR   =  5740121200361040B,
     2    QRL   =  1212121212120000B,
     3    QRL   =  5500121200441740B,
     4    QRR   =  1212121212120000B,
     5    QRR   =  5740121200501740B,
```

```
6       PR1   =  5400470000541640B,
7       PR2   =  1212121212240000B)
 CON(PR3   =  5440000000561640B,
1       FC1   =  0012121200000061B,
2       FC2   =  5400666300621540B,
3       ADR1  =  0012121212120000B,
4       ADR2  =  5620646400661540B,
5       BK1   =  1212121212000000B,
6       BK2   =  6251656142000000B,
7       BK3   =  5540000000721440B)

* INTRODUCTION

 CON(PREA  =  0002121212120000B,
1       PRE   =  2300636543430000B,
2       PRE   =  6567714522000061B,
3       PRE   =  6144624365000062B,
4       PRE   =  5161440047510065B,
5       PRE   =  2451004751460067B,
6       PRE   =  3440304600001050B)
 CON(PREB  =  0012121212120000B,
1       PRE   =  6365160071220000B,
2       PRE   =  6143430023510061B,
3       PRE   =  6623655100160063B,
4       PRE   =  6451652222000061B,
5       PRE   =  4446513000610064B,
6       PRE   =  3440446500201010B)
 CON(PREC  =  0012121212120000B,
1       PRE   =  5165222200000000B,
2       PRE   =  0061230061640064B,
3       PRE   =  0062656771450022B,
4       PRE   =  6500235161630065B,
5       PRE   =  6251462423710045B,
6       PRE   =  3440222400401750B)

* JUMP TABLE TITLE

 CON(TITLE=  6400412444474022B,
1       TI    =  6527656324234065B,
2       TI    =  6243650046664000B,
3       TI    =  3650236101065340B,
4       TI    =  0066462451004000B,
5       TI    =  0000004361224023B,
6       TI    =  4465220000004000B,
7       TI    =  3050237101165320B)
 CON(TI    =  0000000023464000B,
1       TI    =  4644000000004000B,
```

```
 2      TI    = 35606651012653l0B,
 3      TI    = 632423656400400OB,
 4      TI    = 000000006527406 5B,
 5      TI    = 656324236564400OB,
 6      TI    = 304065270134531OB)

* JUMP TABLE OUTPUT BUF

   CON(JT     = 001212121212400OB,
 1      JT1   = 355000000146527OB,
 2      JT2   = 001212121212400OB,
 3      JT3   = 371000000152527OB,
 4      JT4   = 001212121212400OB,
 5      JT5   = 305000000156527OB,
 6      JT6   = 321004000144527OB,
 7      JT7   = 323003000146527OB)
   CON(JT8    = 325002000150527OB,
 1      JT9   = 222300000000400OB,
 2      JT10  = 327043610152527OB)

* MEMORY DUMP OUTPUT BUF

   CON(DP     = 121212121200000OB,
 1      DP1   = 642444470000000OB,
 2      DP2   = 562000000100132OB,
 3      DP3   = 6164645165220022B,
 4      DP4   = 540000001310126OB,
 5      DP5   = 4645236545230022B,
 6      DP6   = 500000631314126OB,
 7      DP7   = 121212121200000OB)
   CON(DP8    = 542000000220120OB,
 1      DP9   = 121212121212000OB,
 2      DP10  = 514012120224120OB,
 3      DP11  = 121212121212000OB,
 4      DP12  = 570012120230120OB)

* INITIALIZE

1AA    SIU1(DEX)          SIL2(DEX)          . STORE INDEX 1,2
       STA(TEMA)          STQ(TEMQ)          . STORE A AND Q
       LDA(ULCON)         STA(OPREG)         . INITIALIZE
       STA(BREAK)         STA(STPINT)        .  CONTROL FLAGS
       LAC(ULCON)         STA(DUMP)          .
       STA(DISJUM)        STA(PRTJUM)        .
       STA(B1FLG)         STA(B2FLG)         .
       STA(B3FLG)         STA(B4FLG)         .
       STA(B5FLG)         STA(B6FLG)         .
```

```
         STA(AFLG)          STA(QFLG)        .
         STA(FCFLG)         STA(ADFLG)       .
         STA(PFLG)          STA(START)       .
         STA(FLOW)          ENI1(0)          .
         EXF(77504B)        ISK1(10)         .  INITIALIZE
         EXF(77521B)        ISK1(10)         .   LIGHTS
         EXF(77441B)        ISK1(10)         .
         EXF(77421B)        ISK1(10)         .
         EXF(77410B)        ISK1(10)         .
         EXF(77404B)        ISK1(10)         .
         EXF(77203B)        ISK1(10)         .
         EXF(77303B)        ISK1(10)         .
         EXF(77205B)        ISK1(10)         .
         EXF(77305B)        ISK1(10)         .
         EXF(77211B)        ISK1(10)         .
         EXF(77311B)        ISK1(10)         .
         EXF(77221B)        ISK1(10)         .
         EXF(77321B)        ISK1(10)         .
         EXF(77241B)        ISK1(10)         .
         EXF(77341B)        ISK1(10)         .
         EXF(77403B)        ISK1(10)         .
         EXF(77503B)        ISK1(10)         .
         EXF(77511B)        ISK1(10)         .
         EXF(77540B)        ISK1(10)         .
         EXF(77010B)        LDA(ADCRESS)     .
         STA(PREG)          ENI2(1)          .  INITIALIZE PSUEDO P-REGISTER
         LDA(ULCON)         STA(UPLOW)       .  INITIALIZE UPPER-LOWER COUNTER
         SLJ4(1E)           ZRO(0)           .

*  CONTROL

1A       LDA7(PREG)         ENI(0)           .  LOAD NEXT INST FROM PROGRAM
         SSH(UPLOW)         ALS(24)          .
         SCL(MSK1)          STA(TEM)         .  STORE IN UPPER HALF OF TEM
         ALS(6)             STA(TM)          .  STORE FUNCTION CODE IN TM
         LIL1(TM)           LDA1(1L)         .  LOOK IN TABLE
         SAL(L+2)           LIU1(TEM)        .
         LDA(UPLOW)         STA(UL)          .
         SIL1(ADDRES)       SLJ(N)           .  SAVE ADD, GO TO EXEC ROUTINE

*  TABLE

1L       ZRO(0)             SLJ(10)          .  ZRO
         ZRO(0)             SLJ(10)          .  ARS
         ZRO(0)             SLJ(10)          .  QRS
         ZRO(0)             SLJ(10)          .  LRS
         ZRO(0)             SLJ(10)          .  ENQ
```

```
ZRO(0)          SLJ(10)          . ALS
ZRO(0)          SLJ(10)          . QLS
ZRO(0)          SLJ(10)          . LLS
ZRO(0)          SLJ(10)          . ENA
ZRO(0)          SLJ(10)          . INA
ZRO(0)          SLJ(10)          . LDA
ZRO(0)          SLJ(10)          . LAC
ZRO(0)          SLJ(10)          . ADD
ZRO(0)          SLJ(10)          . SUB
ZRO(0)          SLJ(10)          . LDQ
ZRO(0)          SLJ(10)          . LQC
ZRO(0)          SLJ(10)          . STA
ZRO(0)          SLJ(10)          . STQ
ZRO(0)          SLJ(1J)          . AJP
ZRO(0)          SLJ(1J)          . QJP
ZRO(0)          SLJ(10)          . MUI
ZRO(0)          SLJ(10)          . DVI
ZRO(0)          SLJ(10)          . MUF
ZRO(0)          SLJ(10)          . DVF
ZRO(0)          SLJ(10)          . FAD
ZRO(0)          SLJ(10)          . FSB
ZRO(0)          SLJ(10)          . FMU
ZRO(0)          SLJ(10)          . FDV
ZRO(0)          SLJ(10)          . SCA
ZRO(0)          SLJ(10)          . SCQ
ZRO(0)          SLJ(1S)          . SSK
ZRO(0)          SLJ(1S)          . SSH
ZRO(0)          SLJ(10)          . SST
ZRO(0)          SLJ(10)          . SCL
ZRO(0)          SLJ(10)          . SCM
ZRO(0)          SLJ(10)          . SSU
ZRO(0)          SLJ(10)          . LDL
ZRO(0)          SLJ(10)          . ADL
ZRO(0)          SLJ(10)          . SBL
ZRO(0)          SLJ(10)          . STL
ZRO(0)          SLJ(10)          . ENI
ZRO(0)          SLJ(10)          . INI
ZRO(0)          SLJ(10)          . LIU
ZRO(0)          SLJ(10)          . LIL
ZRO(0)          SLJ(1S)          . ISK
ZRO(0)          SLJ(1J)          . IJP
ZRO(0)          SLJ(10)          . SIU
ZRO(0)          SLJ(10)          . SIL
ZRO(0)          SLJ(10)          . SAU
ZRO(0)          SLJ(10)          . SAL
ZRO(0)          SLJ(10)          . INT
ZRO(0)          SLJ(10)          . OUT
```

```
        ZRO(0)          SLJ(1S)         . EQS
        ZRO(0)          SLJ(1S)         . THS
        ZRO(0)          SLJ(1S)         . MEQ
        ZRO(0)          SLJ(1S)         . MTH
        ZRO(0)          SLJ(10)         . RAD
        ZRO(0)          SLJ(10)         . RSB
        ZRO(0)          SLJ(10)         . RAO
        ZRO(0)          SLJ(10)         . RSO
        ZRO(0)          SLJ(1S)         . EXF
        ZRO(0)          SLJ(1J)         . SLJ
        ZRO(0)          SLJ(1J)         . SLS
        ZRO(0)          SLJ(10)         . SEV

*  JUMP INSTRUCTIONS

1J      LDQ(MSK2)       LDL(TEM)        . LOAD JUMP INST
        LQC(MSK2)       ADL(1EXEC)      . MOD ADDRESS
        STA(1EXEC)      ENI(0)          .
        LDA(BREAK)      AJP3(L+3)       . IS BREAKPOINT SET
        LDA(BRKPT)      LDQ(MSK5)       .
        MEQ(PREG)       SLJ(L+2)        . CHECK BREAKPOINT
        SLJ4(1D)        ENI(0)          . GO TO DISPLAY
        LDQ(TEMQ)       LDA(TEMA)       . LOAD A AND Q
        SIU1(STORE)     SIL2(STORE)     . SAVE TRACE INDEX 1,2
        LIU1(DEX)       LIL2(DEX)       . LOAD PROGRAM INDEX 1,2
1EXEC   SLJ(L+1)        SLJ(4J)         . EXEC INST  NO JUMP GO TO 4J
        ENI(0)          ENI(0)          .
        SIU1(DEX)       SIL2(DEX)       . SAVE INDEX 1,2
        LIU1(STORE)     LIL2(STORE)     .
        LDA(MSK3)       LDQ(MSK3)       .
        MEQ(TEM)        SLJ(2J)         . IF NOT RET JUMP GO TO 2J
        LDA(PREG)       INA(1)          . PROCESS RET JUMP
        SAU7(ADDRES)    LDA(UPLOW)      .
        AJP2(L+1)       ALS(1)          .
        STA(UPLOW)      SLJ(3J)         .
2J      LDA(UPLOW)      ENI(0)          . PROCESS JUMP
        AJP3(L+1)       ALS(1)          .
        STA(UPLOW)      ENI(0)          .
3J      LDA(PREG)       STA(TEMP)       . SAVE LAST INST ADDRESS
        LDA(ADDRES)     STA(PREG)       . CHANGE P-REGISTER
        SSK(PRTJUM)     SLJ(1T)         . GO TABLE JUMPS
        SSK(DISJUM)     SLJ(1T)         . GO TABLE JUMPS
        SLJ(1A)         ZRO(0)          .
4J      SIU1(DEX)       SIL2(DEX)       . PROCESS NON-JUMP
        LIU1(STORE)     LIL2(STORE)     .
        SSK(UPLOW)      SLJ(1A)         .
        RAO(PREG)       SLJ(1A)         . INCREASE P-REGISTER
```

```
* SKIP INSTRUCTIONS

1S     SSK(UPLOW)       SLJ(10)          . IF UPPER INST GO TO 10
       LDA7(PREG)       SCL(MSK4)        . IF LOWER INST
       ADD(PASS)        STA(2EXEC)       .   MOD 2EXEC
       SLJ(30)          ZRO(0)           .

* OTHER INSTRUCTIONS

20     ZRO(0)           SLJ(40)          .
10     LQC(MSK1)        LDL(TEM)         .
       LDQ(MSK1)        ADL(20)          .
       STA(2EXEC)       ENI(0)           . MOD 2EXEC
30     LDA(BREAK)       AJP2(L+3)        . IS BREAKPOINT SET
       SSK(STPINT)      SLJ4(1D)         . IS STOP ON EACH INST. FLAG SET
       SLJ(L+4)         ZRO(0)           .
       LDA(BRKPT)       LDQ(MSK5)        .
       MEQ(PREG)        SLJ(L+2)         . CHECK BREAKPOINT
       SLJ4(1D)         ZRO(0)           . GO TO DISPLAY
       SIU1(STORE)      SIL2(STORE)      . SAVE TRACE INDEX 1,2
       LIU1(DEX)        LIL2(DEX)        . LOAD PROG INDEX 1,2
       LDQ(TEMQ)        LDA(TEMA)        .
2EXEC  ZRO(0)           ZRO(0)           . EXEC INST, NO SKIP GO TO 40
       STA(TEMA)        STQ(TEMQ)        . PROCESS SKIP
       ENI(0)           LDA(UPLOW)       .
       SIU1(DEX)        SIL2(DEX)        .
       LIU1(STORE)      LIL2(STORE)      .
       AJP3(L+1)        ALS(1)           .
       STA(UPLOW)       SLJ(50)          .
40     STA(TEMA)        STQ(TEMQ)        . PROCESS NON SKIP
       SIU1(DEX)        SIL2(DEX)        .
       LIU1(STORE)      LIL2(STORE)      .
       SSK(UPLOW)       SLJ(1A)          . IF LOWER INST
50     RAO(PREG)        SLJ(1A)          .   INCR P-REGISTER

* DISPLAY OPERATIONAL REGISTERS

1D     SLJ(N)           ENI(0)           . EXIT/ENTRY
       SSK(5E)          SLJ(1D)          . SKIP DISPLAY IF DD65 NOT AVAIL
       ENA(L)           SUB(PREG)        . IF ABOVE OR BELOW
       AJP3(1D)         ENA(20000B)      .   MAIN PROGRAM
       SUB(PREG)        AJP2(1D)         .   SKIP DISPLAY
       SSK(OPREG)       SLJ(10D)         . SKIP IF NOT SELECTED
       SIL2(STORE)      STA(HOLD)        . PREPARE CONTENTS
       STA(HOLD1)       LDA(DEX)         .   OF INDEX REGISTERS,
       STA(HOLD2)       ARS(24)          .   P REGISTER AND
```

```
      STA(HOLD1)        SIL3(HOLD3)       .   EXECUTION ADDRESS
      SIL4(HOLD4)       SIL5(HCLD5)       .   FOR DISPLAY
      SIL6(HOLD6)       LDA(PREG)         .
      STA(HOLD7)        LDA(ADDRES)       .
      STA(HOLD8)        ENI2(1)           .
2D    LDQ(MSK6)         LDL2(HOLD)        . PUT REGISTER
      ENI1(3)           ENI(0)            .   CONTENTS IN
      ALS(3)            QRS(3)            .     BCD FORMAT
      ADL2(HOLD)        IJP1(L-1)         .
      LDQ(MSK7)         STA(HOLD)         . REPLACE ZEROS WITH
      ENI1(4)           ENI(0)            .   THEIR EQUIVALENT
3D    ENA(0)            QRS(6)            .     BCD CODE
      MEQ(HOLD)         SLJ(L+2)          .NO ZERO
      LDL(ZERO)         RAD(HOLD)         . ZERO
      IJP1(3D)          LDA(HOLD)         .
      ALS(12)           STA2(HOLD)        .
      ISK2(8)           SLJ(2D)           .
      ENI2(1)           ENI1(0)           .
4D    LDA2(HOLD)        STA1(DCTB)        . STORE IN OUTPUT BUF
      INI1(2)           ENI(0)            .
      ISK2(6)           SLJ(4D)           .
      LDA(TEMA)         STA(HOLD2)        . PREPARE CONTENTS
      ALS(24)           STA(HOLD1)        .   OF A AND Q
      LDQ(TEMQ)         STQ(HOLD4)        .   REGISTERS
      QLS(24)           STQ(HOLD3)        .   FOR DISPLAY
      ENI2(1)           EXF(77010B)       .
5D    LDQ(MSK8)         LDL2(HOLD)        . PUT REGISTER
      ENI1(6)           ENI(0)            .   CONTENTS IN
      ALS(3)            QRS(3)            .     BCD FORMAT
      ADL2(HOLD)        IJP1(L-1)         .
      LDQ(MSK9)         STA(HOLD)         . REPLACE ZEROS WITH
      ENI1(7)           ENA(0)            .   THEIR EQUIVALENT
      SLJ(L+2)          ZRO(0)            .     BCD CODE
6D    ENA(0)            QLS(6)            .
      MEQ(HOLD)         SLJ(L+2)          . NO ZERO
      LDL(ZERO)         RAD(HOLD)         . ZERO
      IJP1(6D)          LDA(HOLD)         .
      ENI(0)            STA2(HOLD)        .
      ISK2(4)           SLJ(5D)           .
      ENI2(1)           ENI1(0)           .
7D    LDA2(HOLD)        SCL(MSK11)        . STORE
      ALS(12)           STA1(ARL1)        .
      INI1(2)           ENI(0)            .
      ISK2(4)           SLJ(7D)           .   IN
      ENI2(1)           ENI1(1)           .
8D    LDQ(MSK11)        LDL2(HCLD)        .
      ARS(12)           LDQ(MSK10)        .     OUTPUT
```

```
        ADL1(ARL1)      STA1(ARL1)      .
        INI1(2)         ENI(0)          .
        ISK2(4)         SLJ(8D)         .      BUFFER
        LDA(HOLD7)      ALS(6)          .
        STA(PR2)        LDA(UL)         . CHECK FOR
        AJP2(L+2)       LDA(LOW)        .   LOWER OR
        RAD(PR2)        SLJ(L+2)        .     UPPER INST
        LDA(UP)         RAD(PR2)        . STORE P REG
        LDA(HOLD8)      STA(ADR1)       .
        LDQ(MSK12)      LDL(TEM)        . PREPARE FUNCTION CODE
        ARS(3)          QLS(3)          .   FOR DISPLAY
        ADL(TEM)        ARS(3)          .
        QLS(3)          ADL(TEM)        .
        ARS(9)          INA(61B)        .
        SCL(MSK2)       STA(FC1)        .
        LDQ(MSK7)       ENI1(2)         . REPLACE ZEROS
        QRS(6)          ENA(0)          .   WITH THEIR
        MEQ(FC1)        SLJ(L+2)        .     EQUIVALENT
        LDL(ZERO)       RAD(FC1)        .       BCD CODE
        QLS(6)          ENA(0)          .
9D      IJP1(L-3)       ENI1(32)        .
        OUT1(DDTB)      LIL2(STORE)     . OUTPUT TO DD65

* DISPLAY JUMP TABLE

10D     SSK(DISJUM)     SLJ(L+2)        .
        SLJ(1K)         ZRO(0)          .
        LDA(L1)         LIL1(L1)        .
        SAU(10DB)       SIL3(HCLD)      .
10DA    ENI3(3)         ENI(0)          .
        LDQ(MSK6)       LDL1(BUF1)      . PUT JUMP LOCATION
        QRS(3)          ALS(3)          .  IN BCD FORMAT
        ADL1(BUF1)      IJP3(L-1)       .
        STA(JT)         LDQ(MSK7)       .
        ENI3(4)         ENI(0)          .
        ENA(0)          QRS(6)          .
        MEQ(JT)         SLJ(L+2)        .
        LDL(ZERO)       RAD(JT)         .
        IJP3(L-3)       ENI3(3)         .
        LDQ(MSK6)       LDL1(BUF2)      . PUT JUMP DESTINATION
        QRS(3)          ALS(3)          . IN BCD FORMAT
        ADL1(BUF2)      IJP3(L-1)       .
        STA(JT2)        LDQ(MSK7)       .
        ENI3(4)         ENI(0)          .
        ENA(0)          QRS(6)          .
        MEQ(JT2)        SLJ(L+2)        .
        LDL(ZERO)       RAD(JT2)        .
```

```
      IJP3(L-3)       ENI3(3)         .
      LDQ(MSK6)       LDL1(BUF3)      . PUT TIMES EXECUTED
      QRS(3)          ALS(3)          . IN BCD FORMAT
      ADL1(BUF3)      IJP3(L-1)       .
      STA(JT4)        LDQ(MSK7)       .
      ENI3(4)         ENI(0)          .
      ENA(0)          QRS(6)          .
      MEQ(JT4)        SLJ(L+2)        .
      LDL(ZERO)       RAD(JT4)        .
      IJP3(L-3)       LDA(JT)         .
      ALS(12)         INA(4000B)      .
      STA(JT)         LDA(JT2)        .
      ALS(12)         INA(4000B)      .
      STA(JT2)        LDA(JT4)        .
      ALS(12)         INA(4000B)      .
      STA(JT4)        ENA1(0)         .
      STA(HOLD1)      ENA(127B)       .
      ENQ(777B)       STQ(HOLD2)      .
      SUB(HOLD1)      ALS(3)          .
      STA(HOLD1)      LDA(JT1)        .
      SCL(HOLD2)      SCL(MSK15)      .
      ADL(HOLD1)      STA(JT1)        .
      LDA(JT3)        SCL(HOLD2)      .
      SCL(MSK15)      ADL(HOLD1)      .
      STA(JT3)        LDA(JT5)        .
      SCL(HOLD2)      SCL(MSK15)      .
      ADL(HOLD1)      STA(JT5)        .
      ENA(2)          STA(HOLD8)      .
      LDA(HOLD1)      INA(-450B)      .
      ENQ(0)          STA(HOLD1)      .
      DVI(HOLD8)      RAD(HOLD1)      .
      ENA(1306B)      SUB(HOLD1)      .
      STA(HOLD1)      ALS(12)         .
      RAD(JT1)        ENA(4)          .
      RAD(HOLD1)      ALS(12)         .
      RAD(JT3)        ENA(4)          .
      RAD(HOLD1)      ALS(12)         .
      RAD(JT5)        ENI3(6)         .
      EXF(77010B)     OUT3(JT)        .
10DB  ISK1(N)         SLJ(10DA)       . PREPARE TAGS OF LAST
      ENA(127B)       SUB(L4)         . FOUR JUMPS FOR OUTPUT
      ALS(3)          LQC(HOLD2)      .
      SCL(MSK13)      ADL(JT6)        .
      STA(JT6)        ENA(127B)       .
      SUB(L3)         ALS(3)          .
      SCL(MSK13)      ADL(JT7)        .
      STA(JT7)        ENA(127B)       .
```

```
        SUB(L2)           ALS(3)          .
        SCL(MSK13)        ADL(JT8)        .
        STA(JT8)          ENA(127B)       .
        SUB(L1)           ALS(3)          .
        SCL(MSK13)        ADL(JT10)       .
        STA(JT10)         ENI3(5)         .
        OUT3(JT6)         LIL3(HOLD)      .
        SLJ(1K)           ZRO(0)          .

* MEMORY DISPLAY

11D     EXF(77010B)       ENI(0)          .
        ENA(1000B)        STA(ERASE)      .  ERASE
        ENI1(52B)         ENI(0)          .   OPERATIONAL
        OUT(ERASE)        ENA(20000B)           REGISTER
        RAD(ERASE)        IJP1(L-1)               DISPLAY
        ENI1(4)           OUT1(DP3)          DISPLAY TABLE HEADINGS
        ENA(200B)         STA(HOLD1)
        ENA(1320B)        ALS(12)         .
        STA(HOLD4)        ENA(1324B)      .
        ALS(12)           STA(HOLD5)      .
        ENA(1330B)        ALS(12)         .
        STA(HOLD6)        SIL3(HOLD7)     .
        ENI3(17B)         ENI(0)          .
12D     LDA(DPT)          SAU(L+1)        .
        LDA(N)            STA(HOLD)       .  PREPARE CONTENTS
        ARS(24)           STA(HOLD2)      .   OF DUMP ADDRESS
        ENI2(0)           ENI(0)          .     FOR DISPLAY
13D     LDQ(MSK8)         LDL2(HOLD)      .
        ENI1(6)           ENI(0)          .  PUT IN
        ALS(3)            QRS(3)          .   BCD FORMAT
        ADL2(HOLD)        IJP1(L-1)       .
        LDQ(MSK9)         STA(HOLD)       .
        ENI1(7)           ENA(0)          .
        SLJ(L+2)          ZRO(0)          .
14D     ENA(0)            QLS(6)          .  REPLACE ZEROS
        MEQ(HOLD)         SLJ(L+2)        .   WITH THEIR
        LDL(ZERO)         RAD(HOLD)       .     EQUIVALENT
        IJP1(14D)         LDA(HOLD)       .       BCD CODE
        SCL(MSK11)        ALS(12)         .
        STA2(DP9)         LDQ(MSK11)      .  STORE IN
        LDL(HOLD)         ARS(12)         .   OUTPUT
        LDQ(MSK13)        ADL2(DP10)      .     BUFFER
        ADD(HOLD1)        SCL(MSK15)      .
        STA2(DP10)        INI2(1)         .
        ISK2(3)           SLJ(13D)        .
        ENI1(3)           LDQ(MSK6)       .
```

```
         LDL(DPT)        ENI(0)        .  PREPARE DUMP
         ALS(3)          QRS(3)        .   ADDRESS FOR
         ADL(DPT)        IJP1(L-1)     .    DISPLAY
         STA(HOLD3)      LDQ(MSK7)     .
         ENI1(4)         ENI(0)        .
         ENA(0)          QRS(6)        .
         MEQ(HOLD3)      SLJ(L+2)      .
         LDL(ZERO)       RAD(HOLD3)    .
         IJP1(L-3)       LDA(HOLD3)    .
         ALS(12)         STA(DP7)      .
         LDQ(MSK13)      LDL(DP8)
         SCL(MSK15)      ADD(HOLD1)    .
         ADD(HOLD4)      STA(DP8)      .
         LDA(HOLD5)      RAD(DP10)     .
         LDA(HOLD6)      RAD(DP12)     .
         ENI1(6)         OUT1(DP7)     .  OUTPUT TO DD65
         ENA(14B)        ALS(12)       .
         STA(HOLD8)      RAD(HOLD4)    .
         LDA(HOLD8)      RAD(HOLD5)    .
         LDA(HOLD8)      RAD(HOLD6)    .
         LDA(HOLD1)      INA(-20B)     .
         SCL(MSK13)      SCL(MSK4)     .
         STA(HOLD1)      RAD(DPT)      .
         IJP3(12D)       ENI(0)        .
         LIL3(HOLD7)     LIL2(STORE)   .

* KEYBOARD 2

1K       EXF7(77175B)    SLJ(1K1)      .  SENSE KYBD 2 HIT
         EXF(77120B)     INT(KYBD2)    .  INPUT KYBD 2 HIT
         ENQ(40B)        LDL(KYBD2)    .  PROCESS KYBD 2 HIT
         AJP1(1K)        ENQ(10B)      .
         LDL(KYBD2)      ENQ(7B)       .
         AJP(L+1)        ENA(5B)       .
         ADL(KYBD2)      ENQ(20B)      .
         STA(HOLD)       ENA(0)        .
         MEQ(KYBD2)      SLJ(L+4)      .
         LDA(HOLD)       INA(2K)       .
         SAU(L+1)        ENI(0)        .
         LIU1(N)         SLJ(L+4)      .
         LDA(HOLD)       INA(2K)       .
         SAU(L+1)        ENI(0)        .
         LIL1(N)         ENI(0)        .
         SIU1(L+1)       ENI(0)        .
2K       SLJ(N)          ZRO(0)        .  GO TO SELECTED CONTROL FN

* TABLE
```

```
         SLJ(13K)        SLJ(12K)        . B1 - P
         SLJ(14K)        SLJ(6K)         . B3 - PRINT JUMPS
         SLJ(15K)        SLJ(4K)         . B5 - DISPLAY JUMPS
         SLJ(16K)        SLJ(5K)         . A - MEM DUMP
         SLJ(17K)        SLJ(9K)         . FC - STOP ON EA INST
         SLJ(18K)        SLJ(11K)        . B2 - START OVER
         SLJ(19K)        SLJ(3K)         . B4 - DISPLAY OP REGS
         SLJ(20K)        SLJ(8K)         . B6 - DISPLAY FLOW DIA
         SLJ(21K)        SLJ(7K)         . Q - SET BREAK
         SLJ(22K)        SLJ(10K)        . ADD - CONT

* CONTROL FLAGS

3K       SSH(OPREG)      SLJ(L+2)        . DISPLAY OPERATIONAL REGS FLAG
         EXF(77505B)     SLJ(1K)         .
         EXF(77504B)     SLJ4(23K)       .
         SLJ(1K)         ZRO(0)          .
4K       SSH(DISJUM)     SLJ(4KA)        . DISPLAY JUMP TABLE FLAG
4KAA     ENI(0)          EXF(77010B)     .
         ENI1(15)        OUT1(TITLE)     .
         SIL3(HOLD)      LDA(STORE)      .
         SAU(10DB)       ENI1(0)         .
4KAB     EXF(77410B)     SLJ(10CA)       .
4KA      EXF(77411B)     ENI1(500B)      .
         ENA(1061B)      ALS(9)          . ERASE
         STA(ERASE)      EXF(77010B)     . JUMP
         OUT(ERASE)      ENA(20000B)     .  TABLE
         RAD(ERASE)      IJP1(L-1)       .    DISPLAY
         SLJ(1K)         ZRO(0)          .
5K       SSH(DUMP)       SLJ(L+3)        . MEMORY DUMP FLAG
         SLJ4(23K)       ZRO(0)          .
         SLJ(1D+1)       ZRO(0)          .
         SLJ4(23K)       ZRO(0)          .
         SSH(DUMP)       ENA(0)          .
         STA(DPT)        STA(DP)         .
         ENA(4)          STA(BRKDEX)     .
         EXF(77420B)     SLJ(1K)         .
6K       SSH(PRTJUM)     SLJ(L+2)        . PRINT JUMP TABLE FLAG
         EXF(77404B)     SLJ(1K)         .
         EXF(77405B)     SLJ(1K)         .
7K       SSH(BREAK)      SLJ(L+5)        . BREAKPOINT FLAG
         SLJ4(23K)       ZRO(0)          .
         EXF(77520B)     SSH(BREAK)      .
         ENA(4)          STA(BRKDEX)     .
         SLJ(1K)         ZRO(0)          .
         SLJ4(23K)       ZRO(0)          .
```

```
        SLJ(1K)           ZRO(0)            .
8K      ENA(1AA)          EXF(77510B)       . RSDUMP FLAG
        INA(-4)           STA(NAN)          .
        LDA(ADDRESS)      STA(NL)           .
        CALL RSDUMP(NL,NAN,3)
        EXF(77511B)       SLJ(1K)           .
9K      SSH(STPINT)       SLJ(L+3)          . STOP ON EA INST FLAG
        SLJ4(23K)         ZRO(0)            .
        EXF(77440B)       SLJ(1K)           .
        EXF(77441B)       SLJ(1K)           .
10K     SLJ(1D)           ZRO(0)            . CONTINUE
11K     SSH(START)        SLJ(L+2)          . START OVER FLAG
        EXF(77503B)       SLJ(1K)           .
        SLJ4(23K)         ZRO(0)            .
        SSH(START)        EXF(77502B)       .
        SLJ(1K)           ZRO(0)            .
12K     SSH(PFLG)         SLJ(L+2)          . P-REG FLAG
        EXF(77403B)       SLJ(1K)           .
        SLJ4(23K)         ZRO(0)            .
        SSH(PFLG)         ENA(4)            .
        STA(BRKDEX)       EXF(77402B)       .
        SLJ(1K)           ZRO(0)            .
13K     SSH(B1FLG)        SLJ(L+2)          , B1 FLAG
        EXF(77203B)       SLJ(1K)           .
        SLJ4(23K)         ZRO(0)            .
        SSH(B1FLG)        ENA(4)            .
        STA(BRKDEX)       EXF(77202B)       .
        SLJ(1K)           ZRO(0)            .
14K     SSH(B3FLG)        SLJ(L+2)          . B3 FLAG
        EXF(77205B)       SLJ(1K)           .
        SLJ4(23K)         ZRO(0)            .
        SSH(B3FLG)        ENA(4)            .
        STA(BRKDEX)       EXF(77204B)       .
        SLJ(1K)           ZRO(0)            .
15K     SSH(B5FLG)        SLJ(L+2)          . B5 FLAG
        EXF(77211B)       SLJ(1K)           .
        SLJ4(23K)         ZRO(0)            .
        SSH(B5FLG)        ENA(4)            .
        STA(BRKDEX)       EXF(77210B)       .
        SLJ(1K)           ZRO(0)            .
16K     SSH(AFLG)         SLJ(L+2)          . A-REG FLAG
        EXF(77221B)       SLJ(1K)           .
        SLJ4(23K)         ZRO(0)            .
        SSH(AFLG)         EXF(77220B)       .
        ENA(0)            STA(BRKDEX)       .
        SLJ(1K)           ZRO(0)            .
17K     SSH(FCFLG)        SLJ(L+2)          . FUNCTION CODE FLAG
```

```
        EXF(77241B)     SLJ(1K)         .
        SLJ4(23K)       ZRO(0)          .
        SSH(FCFLG)      ENA(2)          .
        STA(BRKDEX)     EXF(77240B)     .
        SLJ(1K)         ZRO(0)          .
18K     SSH(B2FLG)      SLJ(L+2)        . B2 FLAG
        EXF(77303B)     SLJ(1K)         .
        SLJ4(23K)       ZRO(0)          .
        SSH(B2FLG)      ENA(4)          .
        STA(BRKDEX)     EXF(77302B)     .
        SLJ(1K)         ZRO(0)          .
19K     SSH(B4FLG)      SLJ(L+2)        . B4 FLAG
        EXF(77305B)     SLJ(1K)         .
        SLJ4(23K)       ZRO(0)          .
        SSH(B4FLG)      ENA(4)          .
        STA(BRKDEX)     EXF(77304B)     .
        SLJ(1K)         ZRO(0)          .
20K     SSH(B6FLG)      SLJ(L+2)        . B6 FLAG
        EXF(77311B)     SLJ(1K)         .
        SLJ4(23K)       ZRO(0)          .
        SSH(B6FLG)      ENA(4)          .
        STA(BRKDEX)     EXF(77310B)     .
        SLJ(1K)         ZRO(0)          .
21K     SSH(QFLG)       SLJ(L+2)        , Q-REG FLAG
        EXF(77321B)     SLJ(1K)         .
        SLJ4(23K)       ZRO(0)          .
        SSH(QFLG)       EXF(77320B)     .
        ENA(0)          STA(BRKDEX)     .
        SLJ(1K)         ZRO(0)          .
22K     SSH(ADFLG)      SLJ(L+2)        . ADDRESS FLAG
        EXF(77341B)     SLJ(1K)         .
        SLJ4(23K)       ZRO(0)          .
        SSH(ADFLG)      ENA(4)          .
        STA(BRKDEX)     EXF(77340B)     .
        SLJ(1K)         ZRO(0)          .

* MASTER DESELECT

23K     SLJ(N)          SSH(B1FLG)      . CLEAR
        ENA(0)          SSH(B2FLG)      .  CONTROL FLAGS
        STA(BRKPT)      SSH(B3FLG)      .  AND BREAKPOINT
        LDA(BK)         SSH(B4FLG)      .
        STA(BK1)        SSH(B5FLG)      .
        ENI1(3)         SSH(B6FLG)      .
        EXF(77010B)     SSH(AFLG)       .
        OUT1(BK1)       SSH(QFLG)       .
        ENA(0)          SSH(FCFLG)      .
```

```
        STA(BIN)        SSH(ADFLG)      .
        LDA(BK)         SSH(PFLG)       .
        STA(BCD)        SSH(START)      .
        STA(UBCD)       SSH(DUMP)       .
        ENA(13101B)     SSH(BREAK)      .  ERASE
        SSH(BREAK)      ALS(9)      )   .   MEMORY
        STA(ERASE)      ENI1(150B)      .    DUMP
        OUT(ERASE)      ENA(20000B)     .     DISPLAY
        RAD(ERASE)      IJP1(L-1)       .
        ENA(15K1)       SAU(14K1)       .
        EXF(77205B)     ISK1(40)        .
        EXF(77403B)     ISK1(40)        .
        EXF(77521B)     ISK1(40)        .
        EXF(77305B)     ISK1(40)        .
        EXF(77321B)     ISK1(40)        .
        EXF(77221B)     ISK1(40)        .
        EXF(77503B)     ISK1(40)        .
        EXF(77203B)     ISK1(40)        .  CLEAR
        EXF(77303B)     ISK1(40)        .   LIGHTS
        EXF(77341B)     ISK1(40)        .
        EXF(77311B)     ISK1(40)        .
        EXF(77211B)     ISK1(40)        .
        EXF(77241B)     ISK1(40)        .
        EXF(77421B)     ISK1(40)        .
        SLJ(23K)        ZRO(0)          .
* KEYBOARD 1
1K1     EXF7(77172B)    SLJ(1K)         .  SENSE KYBD 1 HIT
        EXF(77140B)     INT(KYBD1)      .  INPUT KYBD 1 HIT
        SSK(BREAK)      SLJ(3K1)        .  PROCESS KYBD 1 HIT
        SSK(B1FLG)      SLJ(L+2)        .
        SLJ(4K1)        ZRO(0)          .
        SSK(B2FLG)      SLJ(L+2)        .
        SLJ(5K1)        ZRO(0)          .
        SSK(B3FLG)      SLJ(L+2)        .
        SLJ(6K1)        ZRO(0)          .
        SSK(B4FLG)      SLJ(L+2)        .
        SLJ(7K1)        ZRO(0)          .
        SSK(B5FLG)      SLJ(L+2)        .
        SLJ(8K1)        ZRO(0)          .
        SSK(B6FLG)      SLJ(L+2)        .
        SLJ(9K1)        ZRO(0)          .
        SSK(AFLG)       SLJ(L+2)        .
        SLJ(10K1)       ZRO(0)          .
        SSK(QFLG)       SLJ(L+2)        .
        SLJ(11K1)       ZRO(0)          .
```

```
        SSK(FCFLG)      SLJ(L+2)        .
        SLJ(12K1)       ZRO(0)          .
        SSK(ADFLG)      SLJ(L+2)        .
        SLJ(13K1)       ZRO(0)          .
        SSK(PFLG)       SLJ(L+2)        .
        SLJ(14K1)       ZRO(0)          .
        SSK(START)      SLJ(L+2)        .
        SLJ(1AA)        ZRO(0)          .
        SSK(DUMP)       SLJ(1K)         .
2K1     LDA(BRKDEX)     ADD(BRKDEX)     .
        ADD(BRKDEX)     ENI(0)          .
        SAL(L+4)        SAL(L+3)        . ADD ONE DIGIT OF
        ENQ(70B)        LDL(KYBD1)      .  DUMP ADDRESS
        AJP1(2K1B)      ENQ(7)          .
        LDL(KYBD1)      ALS(N)          .
2K1A    ENQ(77770B)     QLS(N)          .
        ADL(DPT)        STA(DPT)        .
        LDQ(MSK6)       LDL(DPT)        .
        ENI1(3)         ENI(0)          . PUT IN
        ALS(3)          QRS(3)          .  BCD FORMAT
        ADL(DPT)        IJP1(L-1)       .   FOR DISPLAY
        ALS(18)         STA(DP)         .
        ENQ(7700B)      QLS(6)          . REPLACE ZEROS
        QLS(6)          ENA(0)          .  WITH THEIR
        MEQ(DP)         SLJ(L+2)        .   EQUIVALENT
        LDL(ZERO)       RAD(DP)         .    BCD CODE
        ISK1(4)         SLJ(L-3)        .
        LDA(BRKDEX)     AJP(11D)        . GO DUMP
        RSO(BRKDEX)     EXF(77010B)     .
        ENI1(3)         OUT1(DP)        . DISPLAY DUMP ADDRESS
        SLJ(1K)         ZRO(0)          .
2K1B    ENA(0)          SLJ(2K1A)       .
3K1     LDA(BRKDEX)     AJP2(L+3)       .
        SLJ4(23K)       ZRO(0)          .
        SLJ(1K)         ZRO(0)          .
        ADD(BRKDEX)     ADD(BRKDEX)     .
        SAL(L+4)        SAL(L+3)        . MODIFY
        ENQ(70B)        LDL(KYBD1)      . BREAKPOINT
        AJP1(3K1B)      ENQ(7)          .
        LDL(KYBD1)      ALS(N)          .
3K1A    ENQ(77770B)     QLS(N)          .
        ADL(BRKPT)      STA(BRKPT)      .
        LDQ(MSK6)       LDL(BRKPT)      .
        ENI1(3)         ENI(0)          . PUT IN
        ALS(3)          QRS(3)          . BCD FORMAT
        ADL(BRKPT)      IJP1(L-1)       .FOR DISPLAY
        ALS(18)         STA(BK1)        .
```

```
         ENQ(7700B)       QLS(6)           .  REPLACE ZEROS
         QLS(6)           ENA(0)           .    WITH THEIR
         MEQ(BK1)         SLJ(L+2)         .      EQUIVALENT
         LDL(ZERO)        RAD(BK1)         .        BCD CODE
         ISK1(4)          SLJ(L-3)         .
         EXF(77010B)      ENI1(3)          .
         OUT1(BK1)        RSO(BRKDEX)      .  DISPLAY MODIFIED BRKPT
         SLJ(1K)          ZRO(0)           .
3K1B     ENA(0)           SLJ(3K1A)        .
4K1      SLJ4(15K1)       ZRO(0)           .  B1
         LDA(BIN)         SAU(DEX)         .
         SLJ(1D+1)        ZRO(0)           .
5K1      SLJ4(15K1)       ZRO(0)           .  B2
         LDA(BIN)         SAL(DEX)         .
         SLJ(1D+1)        ZRO(0)           .
6K1      SLJ4(15K1)       ZRO(0)           .  B3
         LIL3(BIN)        SLJ(1D+1)        .
7K1      SLJ4(15K1)       ZRO(0)           .  B4
         LIL4(BIN)        SLJ(1D+1)        .
8K1      SLJ4(15K1)       ZRO(0)           .  B5
         LIL5(BIN)        SLJ(1D+1)        .
9K1      SLJ4(15K1)       ZRO(0)           .  B6
         LIL6(BIN)        SLJ(1D+1)        .
10K1     SLJ4(16K1)       ZRO(0)           .  A-REG
         LDA(BIN)         STA(TEMA)        .
         SLJ(1D+1)        ZRO(0)           .
11K1     SLJ4(16K1)       ZRO(0)           .  Q-REG
         LDA(BIN)         STA(TEMQ)        .
         SLJ(1D+1)        ZRO(0)           .
12K1     SLJ(1K)          ZRO(0)           .  FUNCTION CODE
13K1     SLJ4(15K1)       ZRO(0)           .  ADDRESS
         LDA(BIN)         STA(ADDRES)      .
         LDA(BRKDEX)      AJP3(L+2)        .
         SLJ(1D+1)        ZRO(0)           .
         SLJ4(23K)        ZRO(0)           .
         LDA(1A+7)        SAU(L+1)         .
         SLJ(N)           ZRO(0)           .
14K1     SLJ4(15K1)       ZRO(0)           .  P-REG
         LDA(BIN)         STA(PREG)        .
         LDA(BRKDEX)      AJP3(L+2)        .
         SLJ(1D+6)        ZRO(0)           .
         ENA(L+2)         SAU(14K1)        .
         SLJ(L-2)         ZRO(0)           .
         ENQ(4)           SSH(UPLOW)       .
         LDL(KYBD1)       AJP1(L+2)        .  IS UPPER OR LOWER ADDRESS DESIRED
         SSH(UPLOW)       ENI(0)           .
         SLJ4(23K)        ZRO(0)           .
```

```
       SLJ(1A)       ZRO(0)        .

* FIFTEEN BIT REGISTERS

15K1   SLJ(N)        LDA(BRKDEX)   .
       AJP2(L+2)     SLJ4(23K)     .
       SLJ(1K)       ZRO(0)        .
       ADD(BRKDEX)   ADD(BRKDEX)   .
       SAL(L+4)      SAL(L+3)      .
       ENQ(70B)      LDL(KYBD1)    .
       AJP1(15K1B)   ENQ(7)        .
       LDL(KYBD1)    ALS(N)        .
15K1A  ENQ(77770B)   QLS(N)        .
       ADL(BIN)      STA(BIN)      .
       RSO(BRKDEX)   SLJ(15K1)     .
15K1B  ENA(0)        SLJ(15K1A)    .

* 48 BIT REGISTERS

16K1   SLJ(N)        LDA(BRKDEX)   .
       INA(-16)      AJP2(16K1D)   .
       LDA(BRKDEX)   ADD(BRKDEX)   .
       ADD(BRKDEX)   SAL(L+4)      .
       SAL(L+4)      ENQ(70B)      .
       LDL(KYBD1)    ENI(0)        .
       AJP1(16K1B)   ENQ(7)        .
       LDL(KYBD1)    ALS(N)        .
16K1A  ENQ(77770B)   QLS(N)        .
       ADL(BIN)      STA(BIN)      .
       RAO(BRKDEX)   SLJ(16K1)     .
16K1B  ENA(0)        SLJ(16K1A)    .
16K1D  SLJ4(23K)     ZRO(0)        .
       SLJ(1K)       ZRO(0)        .

* JUMP TABLE TABULATION

1T     ENA(L)        SUB(TEMP)     . IF ABOVE OR BELOW
       AJP3(1A)      ENA(20000B)   .   MAIN PROGRAM
       SUB(TEMP)     AJP2(6T)      .   DO NOT TABLE JUMP ADDRESSES
       LDA(TEMP)     LDQ(MSK5)     . S
       LIL2(STORE)   ENI(0)        .  A
2T     MEQ2(BUF1)    SLJ(3T)       .   V
       IJP2(4T)      ENI(0)        .    E
3T     LIL2(STORE)   STL2(BUF1)    .     ADD JUMPED FROM
       LDA(ADDRES)   STL2(BUF2)    .     ADD JUMPED TO
       ENA(1)        STA2(BUF3)    .     NO. OF TIMES EXECUTED
       LDA(L3)       STA(L4)       .
```

```
        LDA(L2)         STA(L3)         .
        LDA(L1)         STA(L2)         .
        SIL2(L1)        INI2(1)         .
        SLJ(5T)         ZRO(0)          .
4T      INI2(1)         LDA2(BUF2)      .
        MEQ(ADDRES)     SLJ(7T)         .
        RAD2(BUF3)      LDA(L3)         .
        STA(L4)         LDA(L2)         .
        STA(L3)         LDA(L1)         .
        STA(L2)         SIL2(L1)        .
        LIL2(STORE)     SLJ(6T)         .
5T      ENA(49)         ENI(0)          . IF BUF FULL
        MTH(STORE)      SLJ(6T)         .
        SLJ(1P)         ENI(0)          .    GO PRINT
6T      ENA(2600B)      ENQ(37600B)     .
        MEQ(PREG)       SLJ(1A)         . IF NOT IN RESIDENT GO TO 1A
        SLJ(1P)         ENI(0)          . IF IN RESIDENT GO PRINT
7T      LDA(TEMP)       SLJ(2T)         .

* PRINT JUMP TABLE

1P      LDA(PRTJUM)     AJP3(2P)        . CHECK PRINT FLAG
        SIL2(NAN)       ENA(0)          .
        PRINT 10
10      FORMAT(//6X,22HFROM       TO      TIMES,/23X,8HEXECUTED,//)
        PRINT 11,(BUF1(I),BUF2(I),BUF3(I),I=1,NAN)
11      FORMAT(5X,O5,4X,O5,4X,O5)
2P      ENA(2600B)      ENQ(37600B)     .
        MEQ(PREG)       SLJ(3P)         .
        EXF(77010B)     ENI(0)          .
        SSK(5E)         SLJ(L+4)        .
        ENI1(2)         OUT1(END)       .
        ENA(L+2)        SAU(1D)         .
        SLJ(1K)         ENI(0)          .
        LDA(PREG)       SAU(L+3)        . RELINQUISH CONTROL
        LIU1(DEX)       LIL2(DEX)       .
        LDQ(TEMQ)       LDA(TEMA)       .
        SLJ(N)          ZRO(0)          .    TO MONITOR
3P      ENI2(1)         ENA(0)          . CLEAR BUFS
        STA2(BUF1)      STA2(BUF2)      .
        STA2(BUF3)      ENI(0)          .
        ISK2(50)        SLJ(L-2)        .
        ENI2(1)         SLJ(1A)         .

* ERASE DD65 AND DISPLAY INTRODUCTION

1E      SLJ(N)          ENI(0)          .
```

```
      EXF7(77010B)     SLJ(4E)          . SENSE IF DD65 AVAIL
      ENI1(512)        ENI(0)           .
2E    OUT(3E)          IJP1(L)          . ERASE DD65
      ENA(-0)          STA(5E)          . SET NEG FLAG FOR DD65 AVAIL
      ENA(0)           STA(L4)          .
      STA(L3)          STA(L2)          .
      STA(L1)          ENI(0)           .
      LDA(ADDRESS)     STA(HOLD1)       . PUT PROGRAM
      ENA(1AA)         INA(-4)          . ADDRESSES IN
      STA(HOLD2)       ENI2(1)          . BCD FORMAT
2EA   LDQ(MSK6)        LDL2(HOLD)       . FOR INTRODUCTION
      ENI1(3)          ENI(0)           .
      ALS(3)           QRS(3)           .
      ADL2(HOLD)       IJP1(L-1)        .
      LDQ(MSK7)        STA(HOLD)        .
      ENI1(4)          ENI(0)           .
2EB   ENA(0)           QRS(6)           .
      MEQ(HOLD)        SLJ(L+2)         .
      LDL(ZERO)        RAD(HOLD)        .
      IJP1(2EB)        LDA(HOLD)        .
      ALS(12)          STA2(HOLD)       .
      ISK2(2)          SLJ(2EA)         .
      LDA(HOLD1)       STA(PREB)        .
      LDA(HOLD2)       STA(PREC)        .
      EXF(77010B)      ENI1(21)         .
      OUT1(PREA)       ENI2(1)          . OUTPUT INTRODUCTION
      ENA(2EC)         SAU(1D)          .
      SLJ(1K)          ZRO(0)           . GO TO KEYBOARD CONTROL
2EC   ENA(L+2)         SAL(4KAB)        .
      SLJ(4KAA)        ZRO(0)           . GO DISPLAY JUMP TABLE TITLE
      ENA(10DA)        SAL(4KAB)        .
      SLJ(1E)          ZRO(0)           .
3E    ZRO(0)           ZRO(0)           .
4E    ENA(0)           STA(5E)          . SET POS FLAG FOR DD65 NOT AVAIL
      SLJ(1E)          ENI(0)           .
5E    ZRO(0)           ZRO(0)           .
      RETURN
      END
      END


..END
```

thesL636
On-line program analysis and diagnosis.
3 2768 002 11769 9
DUDLEY KNOX LIBRARY